WEST Search History

DATE: Thursday, October 16, 2003

Set Name side by side	Query	Hit Count	Set Name result set
	DB=USPT,PGPB,JPAB,EPAB,DWPI,TDBD; PLUR=YES; OP=ADJ		
L12	L11 and (shutdown or (low adj4 activity) or boot or booting)	50	L12
L11	L10 and (disk near8 (hard or drive))	258	L11
L10	L9 and @AD<20000718	806	L10
L9	(connect or connection) near8 ((personal adj4 computer) or PC) near8 (server or provider)	1870	L9
L8	L7 and @AD<20000718	3	L8
L7	(disk adj4 drive) same (protected or firmware) same (content near8 (server or provider))	8	L7
L6	(disk adj4 drive) same firmware same protected same (content near8 (server or provider))	0	L6
L5	L4 and (shutdown or (low adj4 activity))	6	L5
L4	L3 and (boot or booting)	51	L4
L3	L2 and @AD<20000718	151	L3
L2	L1 and firmware	410	L2
L1	(disk near8 (hard or drive)) same (address or pointer or link) same (server or content)	2857	L1

END OF SEARCH HISTORY

L12: Entry 10 of 50 File: USPT Feb 4, 2003

DOCUMENT-IDENTIFIER: US 6516338 B1
** See image for Certificate of Correction **
TITLE: Apparatus and accompanying methods for implementing network servers for use in providing interstitial web advertisements to a client computer

Application Filing Date (1):
19990713

Brief Summary Text (17):
Interstitial web advertising is taught in, e.g., U.S. Pat. Nos. 5,737,619 and 5,572,643 (both of which issued to D. H. Judson but on Apr. 7, 1998 and Nov. 5, 1996, respectively--hereinafter the "Judson" patents). The Judson patents disclose the concept of embedding an advertisement, as an information object, in a web page file in such a manner that the object will remain hidden and not displayed when the file is executed to render the page. Rather than being displayed, the information object is locally cached by the browser during execution of the code for that page. Then, during a transition initiated by user activation of a hotlink to move from that page to a next successive page, i.e., during an interstitial, the browser accesses the advertisement from local cache and displays it until such time as that next successive page is downloaded and rendered. See also, published International patent application WO 97/07656 (to E. Barkat et al and published on Mar. 6, 1997) which teaches the concept of "polite" downloading. Here, a browser, on a local computer (e.g., a client PC) downloads, from a remote advertising system server and ostensibly as a background process, file(s) for a web advertisement only during those intervals when bandwidth utilization of a communication channel (link) connected to the browser is less than a pre-established threshold. Such "polite" downloading is intended to minimally interfere with other communication applications, then executing on the client PC, which will utilize the link. The browser displays the downloaded ad(s) to the user only after the user has not interacted, as detected by a conventional screen saver process, with his(her) PC for a predefined period of time, such as by neither moving a mouse nor depressing a key on a keyboard during that period. The server selects those advertisements for download to the client PC based on a user-ID and preference information of the user, who is then situated at that PC, and configuration information of that PC, which, when a connection is established between the client PC and the server, the client PC uploads to the server. Though the files associated with an interstitial advertisement can be large, these files are advantageously fetched by a client browser during those intervals when otherwise the browser would be idle and hence bandwidth utilization of its network connection would be relatively low. Such "idle times" would occur, in the absence of processing an interstitial ad, after the browser has fully rendered a web page and a user is viewing the page but has not yet clicked on a hotlink to transition to another page. During such an idle time, the browser would simply wait for further user input.

Brief Summary Text (20):
Given these deficiencies, the art teaches a concept of implementing web advertising through using so-called "push" technology. See, e.g., U.S. Pat. No. 5,740,549 (issued to J. P. Reilly et al on Apr. 14, 1998 --hereinafter the "Reilly et al" patent). In essence and as described in the Reilly et al patent, a client PC, through execution of a "push" application program (called "administration manager"), establishes a network connection with an information server, i.e., a "push" web server, typically during off-hours, such as in the late evening or early morning, or at a predefined interval (e.g., every four hours). The information server then downloads, i.e., "pushes", to the administration manager, content files, such as for advertisements and/or other predefined information, that are to be played to the user sometime later. The administration manager, i.e., the "push" application, in turn, stores all the "pushed" content files into a local database (referred to as the "information database") on a local hard disk and, in response to instructions received from the information server,

deletes those previously "pushed" content files which have already been displayed. The administration manager also maintains a user profile, which specifies user preferences as to the specific advertising and/or other information (s)he wants to receive, in the information database. As such, through each connection, the information server, by selecting content from its database relative to preferences specified in the user profile, attempts to "push" fresh content to the client PC that is likely to be of interest to the user but without duplicating that which was already displayed. Stored "pushed" content is later displayed, using a data viewer, either on user demand or during those times when the user is not interacting with the system, here too detected by a conventional screen saver procedure.

Brief Summary Text (23):
In addition, "push" application programs continue to increase in size, often considerably, as they provide added capabilities to a user. Downloading and then regularly updating a push application will reduce, sometimes considerably, the amount of disk space available to the user on his(her) client PC. Furthermore, "push" applications rely on periodically "pushing" large quantities of media content from a push server to the client PC and storing that content on the hard disk of that PC pending subsequent display. This content, depending on its volume, can consume inordinate amounts of hard disk space. Furthermore, advertisers have discovered, not surprisingly, that relatively few PC users will undertake any affirmative action, such as by downloading and installing an application program--almost regardless of its size, to receive advertisements and other such information.

Brief Summary Text (33):
In that regard, this new technique should preferably not embed advertising HTML files within a web page. If this could be accomplished, then advantageously such a technique would likely provide considerable economies to advertisers in saved labor, time and cost in terms of both inserting advertisements into web page files, and later changing any of those advertisements. In addition, such a new technique should preferably function in a manner that is substantially, if not totally, transparent to a user and which neither inconveniences nor burdens that user. In particular, this new technique should preferably not require a user to download and install on his(her) PC a separate application program, let alone any update to it, specifically to receive web advertising, or perform any affirmative act, other than normal web browsing, to receive such advertising. Furthermore, this new technique should preferably be platform independent and, by doing so, operate with substantially any web browser on substantially any PC. Also, this new technique, when in use, should preferably not consume excessive hard disk space on a client PC. Moreover, to provide a pleasing "user experience", this new technique should render an ad fully and without any interruptions that might otherwise result from network and/or server congestion. Lastly, this new technique should provide proper accounting to an advertiser by accurately and validly ascertaining user impressions of fully rendered advertisements.

Brief Summary Text (40):
In particular, once the agent is started, the agent politely and transparently downloads, through the client browser and to the browser cache, both media and player files, originating from the advertisement management server, for an advertisement that are needed to fully play content in that advertisement. The agent also monitors a click-stream generated by a user who then operates the browser. In response to a user-initiated action, e.g., a mouse click, which instructs the client browser to transition to a next successive content web page and which signifies a start of an interstitial interval, the agent, if all the media and player files are then resident on the client hard disk, plays the media files, through the browser and during that interstitial interval, directly from the browser cache. Advertisements are interstitially played typically in the order in which they were downloaded to the client browser. Interstitial play from browser cache advantageously permits previously cached content rich advertisements to be played through the browser without adversely affecting communication link bandwidth then available to the client browser. Thus, the full available link bandwidth can be used, while an advertisement is being played, to download a next successive content web page.

Brief Summary Text (43):
At the start of an interstitial interval, the agent determines whether all the media and player files required to play a given advertisement (typically that having its so-called AdDescriptor file situated in a head of a play queue) then reside on the disk of the client PC or, with respect to media files, are resident in browser RAM cache. If so, the agent then accesses these files from the disk to "play" that advertisement. Since all the media and player files are then locally resident, the advertisement, from

a user's perspective, is immediately rendered from the client hard disk or browser RAM cache with essentially no downloading delay, thus providing a highly pleasing "user experience" with rich multi-media content approaching that obtainable through current CD-ROM based delivery. Thereafter, the agent returns control to the browser to permit the browser, if a next successive web page has been downloaded, assembled and ready to be rendered, to render that particular page to the user. If, however, an advertisement is prematurely terminated by a user, that advertisement (in terms of its AdDescriptor file) will remain in a play queue (with its media and player files remaining on the client hard disk or, in the case of media files, in browser RAM cache) and will be re-played from its beginning at the start of a next successive interstitial interval. Furthermore, if download of the media and player files for an advertisement were to be interrupted by a user click-stream, i.e., start of interstitial interval, the agent suspends further downloading until after the ensuing interstitial interval terminates. To conserve communication link bandwidth, the agent then resumes downloading of these files at a point it was suspended, rather than, as conventionally occurs, totally re-starting the download.

Brief Summary Text (47):
The Transition Sensor applet then passes the URL of the ad management system, as specified in the advertising tag, to the AdController applet in order for the latter applet to request delivery of an advertisement, specifically an associated AdDescriptor file, originating from that system. The system then selects the advertisement to be delivered and, via the third party advertising server, so informs the AdController applet by returning the requested AdDescriptor file. For a given advertisement, this particular file, which is textual in nature, contains a manifest, i.e., a list, of: file names and corresponding web addresses of all media files that constitute content for that advertisement and all player files necessary to play all the media files; an order in which the various media files are to be played; and various configuration and other parameters need to configure and operate the operation of each player in order for it to properly play a corresponding media file(s). The AdController then "politely" downloads, typically via the advertising distribution server, the associated media and player files, as specified in the AdDescriptor file--and to the extent they do not already reside on the hard disk of the client PC. As noted above, the Transition Sensor applet also monitors a click-stream produced by the current user to detect a user-initiated page transition and hence the start of an interstitial interval.

Detailed Description Text (9):
Advertising distribution HTTP server (also referred to as "agent" server) 15 is connected, via communications link 17, to Internet 10 and stores files that collectively implement a predefined agent, specifically, a light weight Java applet. This agent (referred to herein as the "AdController" agent) transparently pre-loads itself, as well as media rich advertising content, into a local hard disk cache associated with the browser ("browser disk cache") on client PC 5. Server 15 downloads the AdController agent in a manner to be described below, to client browser 7. This agent, once instantiated and started, then transparently and politely downloads (actually pre-loads) advertisements into the browser disk cache, and subsequently plays each of those advertisements, on an interstitial basis, in response to a click stream generated by the user as (s)he navigates, through use of browser 7, between successive web pages. Such hard disk caching advantageously circumvents variable latency and erratic play associated with conventional streamed and static media delivered over the Internet. The agent enables rich advertising to be presented in a highly-controlled fashion, resulting in user experiences approaching that of CD-ROM.

Detailed Description Text (18):
Advantageously, the agent operates independently, in the client browser, of the content in any referring web page. Once loaded and started, the agent executes in parallel, with standard browser functionality, continually and transparently requesting and downloading advertisements to a browser disk cache residing on a local hard disk ("browser disk cache"), as well as in the case of media files into browser RAM cache, in a client computer (e.g., personal computer--PC) and interstitially playing those advertisements.

Detailed Description Text (29):
Once the AdDescriptor file is downloaded to the client PC, via agent server 15, the AdController then "politely" downloads, as symbolized by block 70 shown in FIGS. 1B and 1C, into the browser disk cache each media and player file, as specified in the AdDescriptor file--to the extent that file does not already reside on the hard disk of the client PC. Through so-called "polite" downloading, media and player files are downloaded to browser 7 during browser idle time intervals, with the downloading

suspended during each ensuing interstitial interval after the user instructs browser 7 to navigate to a new content web page. In this manner, while a fully downloaded advertisement is interstitially played from browser cache, the new content page is downloaded over the full bandwidth of communications link 9. Advantageously, the communications link is freed during each interstitial interval to just carry web page content, thereby expediting download of content pages. If, due to the occurrence of an interstitial interval, the AdController applet suspends downloading of an advertisement file, then upon termination of this interval, this applet then resumes downloading at a location in that file at which downloading had stopped, thus conserving communication bandwidth and reducing download time.

Detailed Description Text (30):
In particular, as part of the operations symbolized by block 70, the AdController applet determines which files, of those listed on the AdDescriptor, do not then reside on the hard disk of client PC 5. Once it has made that determination, this applet issues a request, as symbolized by line 72, to agent server 15, to fetch a first one of these files. The agent server, again serving as a proxy server, issues a request, as symbolized by line 74, to fetch this file from a networked server, anywhere on Internet 10, on which that file resides. For simplicity, we assume that all such files reside on server 20 and are accessible through ad management system 25. Hence, system 25, via server 20, issues a response, as symbolized by line 76 to agent server 15, containing this first advertisement file. The agent server, in turn and as symbolized by line 78, downloads this particular file to client browser 7 for storage in the browser disk cache. Downloading of advertisement files continues in this manner until, as symbolized by line 88, a last required file for the advertisement has been downloaded, via agent server 15, to the browser disk cache on client PC 5.

Detailed Description Text (40):
As shown, the client PC comprises input interfaces (I/F) 320, processor 340, communications interface 350, memory 330 and output interfaces 360, all conventionally interconnected by bus 370. Memory 330, which generally includes different modalities, including illustratively random access memory (RAM) 332 for temporary data and instruction store, diskette drive(s) 334 for exchanging information, as per user command, with floppy diskettes, and non-volatile mass store 335 that is implemented through a hard disk, typically magnetic in nature. Mass store 335 may also contain a CD-ROM or other optical media reader (not specifically shown) (or writer) to read information from (and write information onto) suitable optical storage media. The mass store stores operating system (O/S) 337 and application programs 400; the latter illustratively containing browser 7 (see, e.g., FIGS. 1B and 1C) which implements our inventive technique. O/S 337, shown in FIG. 3, may be implemented by any conventional operating system, such as the WINDOWS NT, WINDOWS 95, or WINDOWS 98 operating system ("WINDOWS NT", "WINDOWS 95" and "WINDOWS 98" are trademarks of Microsoft Corporation of Redmond, Washington). Given that, we will not discuss any components of O/S 337 as they are all irrelevant. Suffice it to say, that the browser, being one of application programs 400, executes under control of the O/S.

Detailed Description Text (47):
As shown, the application programs, to the extent relevant, contain browser 7 and resident JAVA player files 410, i.e., files for JAVA media players that have previously been installed onto the hard disk of the client PC. These players may illustratively include audio, streaming audio, video and multi-media players.

Detailed Description Text (63):
In essence, once Transition Sensor applet 422, as shown in FIG. 5, supplies AdController applet 424 with a URL of an AdDescriptor file, Ad Pipeline 545 then downloads, as symbolized by dot-dashed line 520, the AdDescriptor file, via agent server 15 (serving as a proxy server), from a remote advertising management system. As noted above, this file contains a manifest of media and player files necessary to fully play a complete advertisement. Once this AdDescriptor file has been downloaded into Ad Pipeline 545 then, once the downloading (to the extent needed) is complete, pipeline 545 then "politely" downloads, as symbolized by line 525, each file specified in the manifest--to the extent that file does not already reside on the client hard disk. Pipeline 545 then, once the downloading (to the extent needed) is complete, writes the AdDescriptor file to the play queue and each downloaded file specified therein to browser disk cache 430; hence forming a queued advertisement for subsequent access.

Detailed Description Text (64):
At the inception of an interstitial interval, signaled by a Transition Sensor stop event, the AdController applet interstitially plays an advertisement that has then been

completely queued--both in terms of its media and player files. In particular, at the start of that interval, the Ad Pipeline retrieves an AdDescriptor that is then situated at the head of a play queue. Media players 565 required by that advertisement, as specified in the AdDescriptor file, are started in the order specified in that file along with their corresponding media file(s). A resulting processed media stream, produced by the player(s), and as symbolized by line 570, is rendered through browser 7 to the user. Media players 565 may permanently reside, i.e., apart from being downloaded by agent 420, on the client hard disk (thus being implemented by resident player files 410 as shown in FIG. 4) or be downloaded by pipeline 545 into browser disk cache 430 (and also browser RAM cache) for subsequent access and use (thus stored within files 437 shown in FIG. 4).

Detailed Description Text (67):
Upon entry into process 600 as shown in FIGS. 6A and 6B, which occurs in response to an Transition Sensor init event from browser 7, block 610 is performed. Through this block, Transition Sensor applet 422 instructs the applet registry to load the AdController applet. Once that occurs, block 615 is performed through which external AdController configuration file 620 is retrieved from agent server 15. Thereafter, through decision block 630, agent 420 waits, by looping through NO path 631, until browser 7 generates a Transition Sensor start event. When such an event occurs, execution proceeds, via YES path 633 emanating from this decision block, to block 635. Through this latter block, AdController applet 424 obtains an Internet address of an advertisement management system (e.g., system 25) from which the agent is to retrieve AdDescriptor file 645. Applet 424 then passes this address to Ad Pipeline 545. The Ad Pipeline, as indicated in block 640, then retrieves AdDescriptor file 645 from this address and particularly through agent server 15 serving as a proxy server. Once this file is retrieved, the agent performs block 650 which "politely" downloads all the media and player files 655 (to the extent each file does not already reside on the client hard disk), from advertising management system 25 (residing on advertising server 20), and, through block 660, stores these files into browser disk cache 430 (and, in the case of media files, into browser RAM cache). As noted above, these files are downloaded via agent server 15, which here too serves as a proxy server. This downloading continues until either it finishes or a Transition Sensor stop event generated by the browser arises, whichever occurs first. As to the stop event, decision block 665 tests for its occurrence, with execution looping back, via NO path 666, in the absence of such an event. However, whenever this event occurs, such as (as discussed above) in response to a user-initiated page transition, decision block 665 routes execution, via YES path 668, to block 670. This latter block then, using media players 565, plays an advertisement then fully queued in the play queue on the browser disk cache, i.e., an AdDescriptor file for this ad then resides at a head of the play queue and all associated media and player files for that advertisement, as specified in that AdDescriptor file, then reside on the client hard disk.

Detailed Description Text (72):
Upon entry into operations 800, which occurs in response to an init event produced by the Transition Sensor applet, block 805 is performed. Through this block, the AdController applet is initialized. This includes downloading files, to the extent needed, for this applet from the agent server and then instantiating this applet. Once this occurs, block 810 tests for an occurrence of AdController start event produced by the Transition Sensor applet. Until this event occurs, execution merely loops back, via NO path 812, to block 810. When this event occurs, decision block 810 routes execution, via YES path 814, to block 815. This latter block, retrieves external AdController configuration file 620 from the agent server. Thereafter, block 820 occurs through which the AdController applet creates and starts Ad Pipeline 545. Once the pipeline is fully started, then, block 825 is performed to enable advertisement files to be "politely" downloaded into the Ad Pipeline and to thereafter actually download such files. While advertisement files are being downloaded or thereafter, if such downloading has completed, decision block 830 tests for an occurrence of a Play Ad event. If no such event occurs, then execution loops back, via NO path 833, to decision block 830 to continue any further downloading. If however, a Play Ad event occurs, then decision block 830 routes execution, via YES path 837, to block 840. This latter block suspends further downloading of advertisement files into the Ad Pipeline. Once this occurs, then block 845, when performed, issues a request to the Ad Pipeline to play an advertisement having its AdDescriptor file then located at the head of the play queue. While the advertisement is being played, decision block 850 tests for an occurrence of an shutdown event generated by the browser, such as caused by, e.g., a user-initiated transition or the user closing an advertisement window or closing the browser itself. If such an event does not occur, decision block 850 routes execution, via NO path 853, back to block 825 to re-enable "polite" download of advertisement files once again. If

such a shutdown event occurs, then processing operations 800 terminate, via YES path 857.

Detailed Description Text (103):
Once AdDescriptor file 645 is inserted into the download queue, then block 1440 executes to invoke Ad Downloader process 1700. Process 1700, which will be discussed below in conjunction with FIG. 17, performs a single chain of tasks. First, process 1700 blocks until such time as the downloaded AdDescriptor file has become available in the download queue. During its execution, this process asks download queue 1430 if it contains an AdDescriptor file, e.g., file 645. If so, then advertising files need to be downloaded for that particular AdDescriptor file. If the download queue is empty, then process 1700 both waits until that queue is not empty and also retrieves the AdDescriptor file over the network. Once Ad Downloader process 1700 has obtained this AdDescriptor file, process 1700 then downloads, all the media and required player files specified in the AdDescriptor file by using Browser Cache Proxy 1450, into browser disk (and PAM) cache 1460. Once all the advertising files have finished downloading, process 1700 moves the AdDescriptor file to play queue 1470. However, if the play queue is then full, the Ad Downloader process waits until play queue 1470 is not full before moving the AdDescriptor file into this queue for subsequent ad play. As discussed above, AdDescriptor file 645 for a fully queued ad (i.e., with its all the associated media and player residing on the client hard disk) is subsequently retrieved from play queue 1470 in response to a request to play an advertisement, this request being issued in response to a Transition Sensor stop event.

Detailed Description Text (113):
In particular upon entry into process 1700, execution proceeds to decision block 1710. This decision block determines whether the download queue then contains an AdDescriptor file, e.g., file 645. If the queue is empty, then execution merely loops back, via NO path 1717, to this decision block to await such an AdDescriptor file. However, if download queue 1430 then contains such a file, process 1720 obtains the AdDescriptor file then situated at the head of this queue. Thereafter, block 1730 executes. This block downloads all the required advertising files, not then resident on the client hard disk, into browser proxy cache 1450. This block also transfers all the associated media files in the browser proxy cache to the browser RAM cache. Execution then proceeds to decision block 1740 which determines whether all required advertising files have then been downloaded. If any such file remains to be downloaded, then decision block 1740 routes execution, via NO path 1747, back to block 1730 to download that file. Alternatively, if all the required advertising files have been downloaded, then execution proceeds, via YES path 1743, to block 1750. This latter block moves the AdDescriptor file from download queue 1430 to an end of play queue 1470. Once the AdDescriptor file is written into the play queue, the corresponding advertisement is then ready to be presented to the user, in order relative to other AdDescriptor files then queued in the play queue, during an ensuing interstitial interval.

L12: Entry 15 of 50 File: USPT Mar 5, 2002

DOCUMENT-IDENTIFIER: US 6353848 B1
TITLE: Method and system allowing a client computer to access a portable digital image capture unit over a network

Application Filing Date (1):
19980731

Brief Summary Text (15):
A problem with the above described prior art applications is that access to the Internet and communication thereon require a separate host computer system (e.g., a personal computer system) on each side of the Internet connection in addition to the server computer system on the Internet. The two host computer systems provide the computational resources to host the respective software applications, the Internet access software, and any necessary device drivers. The required computational resources consume a significant amount of memory. Because of this, among other reasons, the above prior art applications are not easily transferred to the realm of easy-to-use, intuitive, consumer electronic devices such as digital cameras, which are small in size and so generally constrained by the amount of memory they can house. In addition, a consumer electronic device such as a digital camera that requires a separate computer system would be more expensive and complex, and therefore would not be consistent with the desire of consumers for lower cost and simpler devices.

Detailed Description Text (14):
Refer now to FIG. 2 which illustrates server computer system 190 upon which embodiments of the present invention may be practiced (the following discussion is also pertinent to a client computer system). In general, server computer system 190 comprises bus 200 for communicating information, processor 201 coupled with bus 200 for processing information and instructions, random access memory 202 coupled with bus 200 for storing information and instructions for processor 201, read-only memory 203 coupled with bus 200 for storing static information and instructions for processor 201, data storage device 204 such as a magnetic or optical disk and disk drive coupled with bus 200 for storing information and instructions, optional display device 205 coupled to bus 200 for displaying information to the computer user, optional alphanumeric input device 206 including alphanumeric and function keys coupled to bus 200 for communicating information and command selections to processor 201, optional cursor control device 207 coupled to bus 200 for communicating user input information and command selections to processor 201, and network interface card (NIC) 208 coupled to bus 200 for communicating from a communication network to processor 201.

Detailed Description Text (22):
RAM disk 532 is a memory area used for storing raw and compressed data and typically is organized in a "sectored" format similar to that of conventional hard disk drives. In the present embodiment, RAM disk 532 uses a well-known and standardized file system to permit external devices, via I/O 448 of FIG. 4, to readily recognize and access the data stored on RAM disk 532. System area 534 typically stores data regarding system errors (for example, why a system shutdown occurred) for use by CPU 444 (FIG. 4) upon a restart of computer 318 (FIG. 3).

Generate Collection

L12: Entry 16 of 50 File: USPT Feb 5, 2002

DOCUMENT-IDENTIFIER: US 6345294 B1
TITLE: Methods and apparatus for remote configuration of an appliance on a network

Abstract Text (1):
A network appliance is capable of remote booting and obtaining its configuration information from a source located far away. The network appliance can be shipped to a business location or office environment without requiring a local boot server in that location or environment and without requiring the presence of a person who is familiar with and highly skilled in configuring the appliance. The invention allows for booting and the obtaining of configuration information, and therefore allows for the functioning of the appliance, regardless of whether there is a local server in the local network environment, such as a DHCP server or a boot server, that has been set up and configured to provide to the appliance the booting and configuration information it requires. Self-organizing distributed appliances (SODAs) according to the invention augment the Internet by providing a self-organizing network that efficiently distributes big data items, i.e., data items that cannot be downloaded timely (on demand) over today's networks. One application of self-organizing distributed appliances is the distribution of high-quality video (a half-hour MPEG-1 movies is about one Gbyte). The SODA network alleviates network bottlenecks.

Application Filing Date (1):
19990419

Brief Summary Text (5):
A typical networking appliance product, which might or might not have a keyboard or monitor and which might not yet have been configured except for some factory standard material incorporated into it, must boot when installed into a local area network, and then a person near that appliance can configure the appliance by filling out forms or typing in parameters into screens or into a terminal.

Brief Summary Text (6):
Alternatively, the appliance, when it is turned on, might send a message out on the local area network asking for any machine in the environment to tell it everything it needs to know in order to be useful, and if there is a boot server configured on the local area network, it will respond to the appliance by giving it some pre-configured information that will help the appliance configure itself. The information required by the appliance might include the name of the appliance, the network address of the appliance, and possibly the pieces of software to be installed in the appliance.

Brief Summary Text (8):
According to certain protocols, a certain kind of message is sent into the local area network to see whether a response is received from a particular kind of server in the local area network. A machine that is in the process of booting might broadcast a boot request packet into the local area network, so that the request could be seen by any computer attached to the immediate local area network (but not by far-away computers).

Brief Summary Text (9):
This well-known technology has been used by many people, companies, and organizations for computers that are booting, even for simple things such as sending network configuration information to a user's personal computer. When a personal computer on a local area network boots up, one of the first things that it might do is to send out a DHCP request message or a boot request message, and a DHCP server or boot server on the local area network, configured by an information systems department, will reply to that request and authoritatively tell the personal computer its network configuration and a set of network parameters to use. This common technique for booting machines that require assistance in booting was employed by Sun Microsystems's network workstations during the mid 1980's.

Brief Summary Text (10):
Today, it is quite common for computers in large organizations to be configured so that they will broadcast a specific kind of request in the form of packet when they are booting to obtain some kind of network configuration information. More specifically, according to a normal protocol for booting an appliance, the appliance, upon booting, uses software to broadcast a message in order to attempt to obtain network configurations and possibly other information required by the appliance. The appliance broadcasts a packet, which may be a bootp (booting protocol) request or a DHCP (dynamic host configuration protocol) request, using well-known technologies (DHCP is a protocol designed for configuring hosts dynamically, which means that the configuration of the host is not stored on that host or appliance itself, but rather is obtained dynamically by sending the DHCP request to a boot server and receiving a response). The appliance receives in response a short message that has a table of settings that include several parameters such as, for example, the IP address of a gateway or router to use when sending packets to far-away places, an IP address to be used as the address of the computer itself, an IP address to be used for sending messages to a web proxy server or HTTP proxy server, addresses of computers that provide naming services (so-called domain name servers), and a bit field that is used as a network mask (which helps indicate to the routing computer which set of addresses are addresses of locally connected computers, thereby allowing the routing computer to distinguish them from addresses of far-away computers). These five or six pieces of information can go a long way in configuring the network communication part of the software on the appliance. This is well-known technology. For example, in every Microsoft Windows 95, 98 or Windows NT computer there is an option in the network control panel that allows a user, rather than specify the IP address of the computer manually by typing into a box, to tell the computer using a dialog check box that every time it boots it should broadcast a message and try to obtain the IP address from a DHCP server. NT servers are provided with a built-in DHCP server that can provide this IP address to other computers upon request.

Brief Summary Text (11):
There are also products being sold into business locations or office environments that, instead of being configured by a boot server, can be configured by a person with a computer such as a laptop that is attached through some kind of cable to configure the appliance. Other products can be configured through use of an LCD panel and buttons. For example, a printer or a photocopying machine, when it is booted up, might display a small message on a screen saying that the user must proceed through menus and select certain options for printer or copier. Similarly, a telefacsimile machine might require a user to set the phone number and the number of rings after which the telefacsimile machine will auto-answer. These configurations are done by a knowledgeable person, and the configuration settings are typically stored within the product itself. If a printer, photocopying machine, or telefacsimile machine is damaged and needs to be replaced, it will be necessary for a knowledgeable person to configure a new machine.

Brief Summary Text (14):
The invention provides a network appliance that is capable of remote booting and is capable of obtaining its configuration information from a source located far away. The network appliance can be shipped to a business location or office environment without requiring a local boot server in that location or environment and without requiring the presence of a person who is familiar with and highly skilled in configuring the appliance.

Brief Summary Text (16):
Also, the invention allows for booting and the obtaining of configuration information, and therefore allows for the functioning of the appliance, regardless of whether there is a local server in the local network environment, such as a DHCP server or a boot server, that has been set up and configured to provide to the appliance the booting and configuration information it requires. Whether there is a boot server or a DHCP server in the local environment or not, the appliance can successfully boot and select a set of network parameters. For example, the appliance can obtain the address of the router, without which the appliance would not be able to communicate into the Internet and could communicate only within the local area network. Furthermore, the appliance is able to communicate with a far-away appliance registry and thereby obtain much more comprehensive configuration information for the appliance. Thus, the invention allows for the easy provision of extensive configuration information to the appliance.

Drawing Description Text (5):
FIG. 4 is a flowchart of the boot procedure of an appliance according to the invention.

Detailed Description Text (16):
remote boot: how to configure a remote appliance without sending a system administrator to the location;

Detailed Description Text (23):
Remote Boot

Detailed Description Text (25):
Referring to FIG. 3, once connected to a LAN 14, the SODA appliance 18 runs a boot algorithm to configure itself. The goal of the boot algorithm is to learn enough about the IP environment in which the appliance is installed to obtain a connection with an appliance registry 28 in order to download additional configuration information. Ideally, the appliance can begin operation without requiring any local user intervention, but if a local administrator must assign some network parameters, it will be most convenient to use nothing more than a web browser and a form page. Only in rare cases should it be necessary for the local installer to install special beacon software to configure an appliance 18. Achieving these goals is challenging because an appliance might, for example, be directly connected to an Ethernet switch or be behind a firewall.

Detailed Description Text (26):
LAN 14 has a router 20 that would normally would connect LAN 14 to the Internet 10. For example, when the user sits at a personal computer 16 and surfs the World Wide Web using a web browser, the web browser on the user's personal computer sends messages that go across LAN 14 into the router 20, which sends the messages across a link 22 that goes from LAN 14 into an Internet service provider 24 that has many connections that go throughout the Internet 10. The messages come out of Internet 10 over at another Internet service provider 26, typically, where there is a connection to a web server that's sending web pages to the user's browser on personal computer 16.

Detailed Description Text (27):
Each appliance 18 is capable of remote booting. When appliance 18 is turned on, it has to boot, and it has to make itself useful after booting. Whereas a typical appliance might boot using some fairly well known protocols, appliance 18 carries out a more sophisticated procedure in order to boot in a variety of different kinds of TCP/IP networking environments and to get configuration information from the owner of the appliance that tells that appliance what kinds of things it is supposed to do.

Detailed Description Text (28):
Appliance 18 is also capable of booting in a place such as an office environment while obtaining its configuration from a source located far away. There can be a great variety of network configurations in a typical office environment, and so the product booting in that environment has to deal with whatever kind of networking environment is present there, and it has to find some way of communicating with the far-away place from which it obtains the configuration information.

Detailed Description Text (29):
Appliance 18 has a remote boot capability and a remote configuration capability that allows the appliance to obtain the information it needs to boot and configure itself from an appliance registry 28, which is located at a remote location. This registry has an attached database 30, which has a number of tables in it, including an ownership table 32, a boot status table 34, and a configuration table 36. These tables are portions of database 30. This database is attached to appliance registry 28, and the appliance registry is a highly available service that can be communicated with through a variety of protocols over Internet 10 and over public access networks. Appendix A describes the contents of database 30.

Detailed Description Text (32):
When appliance 18 boots, it observes the local environment of LAN 14. Appliance 18 broadcasts a request and sees whether there are responses.

Detailed Description Text (34):
With reference to FIG. 4, the appliance, upon being powered on (step 100), makes use of known protocols of bootp or DHCP requests (step 102) to obtain a source of network parameters. The boot server or DHCP server is a computer that acts as a server in the local networking environment and that responds to certain types of route requests messages. A boot server or DHCP server typically responds with a small message that

contains some parameters that the requesting computer needs to be given (step 104). These parameters typically include the IP address of the appliance that is attempting to boot, the subnet mask of the appliance, the IP addresses of one or more routers (typically one router closest to the appliance, such as a router within the same building as the appliance, which may be connected directly to the Internet or which instead may be internal for a large building), one or more name servers (typically two or more name servers; computers, in order to operate properly, often need to be told the address of the name server that is used to translate the names of computers, including addresses of computers), as well as numerous optional parameters.

Detailed Description Text (35):
The appliance can construct candidate network parameters both by communicating with a boot server or DHCP server, as described above, and by simply observing other traffic on the network. If the appliance has received candidate network parameters from a boot server or a DHCP server (step 104), it will test these parameters by attempting to send and receive network messages (step 106).

Detailed Description Text (37):
If the steps described above fail, the appliance assumes that the information obtained from the boot server or DHCP is bad, and the appliance then observes the network passively (step 108). Likewise, if no boot server or DHCP server responds in step 104, the appliance proceeds to step 108. By observing any network traffic on the locally attached network, the appliance is able to build a map of what kinds of network traffic and what network addresses are in use on that network (step 110). In other words, by observing how other computers in the local environment are communicating, the appliance may be able to infer settings that will allow the appliance to communicate in that environment. It is able to deduce typically all of the pieces of information that it would otherwise obtain from a DHCP server or boot server.

Detailed Description Text (46):
Thus, in addition to the ability to reboot without necessarily having a boot server, a key innovation is the above-described technique of observing the network to come up with candidate network parameters, which is followed by testing the parameters by doing some local communication to try to confirm that they are working before moving on to a more difficult communication task such as communicating with the appliance registry.

Detailed Description Text (49):
Thus, the appliance learns some appropriate network settings and algorithms that the appliance can almost certainly use to communicate with the remote appliance registry, including the transmitting of a boot status message to the appliance registry (step 114). The step of transmitting information to the registry can be a one-way communication that the appliance transmits to the registry without the appliance receiving anything back from the registry, or it can be a two-way communication in which the appliance receives a response from the registry.

Detailed Description Text (51):
In the case where there is no boot server or DHCP server, the appliance identifies its own address by selecting a temporary IP address for itself. After having observed the network and tested to see which IP addresses are in use on this network and which ones are in use by specific other computers (step 108), the appliance selects an address that appears not to be in use by any other computer (step 110) and tests it to be more certain that it is not in use (112) and then it uses that address for a very brief period of time as its own address. The appliance uses that address to send messages to the registry (step 114), which can send to the appliance an actual IP address. Because the appliance registry is sending its messages to the appliance from far away on the internet, the messages are sent from the registry to the router that would be trying to deliver that message on the local network (step 116), and at that point, the router, because it doesn't know the correct internet address for the appliance, sends out an ARP message that is received by the appliance (step 118). When the appliance receives the message it responds with its ethernet address (step 120).

Detailed Description Text (54):
Accordingly, a key innovation pertains to the manner in which the appliance sends boot status messages to the appliance registry. Because it may be possible that there is a fire wall 38 (FIG. 3) or proxy server that is screening or preventing some traffic from traveling from local area network 14 into the wide area network, it may be very difficult to communicate with appliance registry 28. Accordingly, the remote booting system is implemented in a manner such that boot status messages (step 114, FIG. 4) can be carried through just about any available network protocol. For example, they can be

carried via e-mail, by domain name service look-up request, by IP packets, and over HTTP requests. This technique is important because there is in almost every network environment that is used to communicate with the internet at least some communication path that will allow the appliance to communicate with the appliance registry. There might be no such communication path if the appliance is operating in a building that is deliberately a secure facility in which it is desired that there be no communication with the Internet. Other than those situations, however, in virtually every common commercial situation there will be some way of communicating with the appliance registry, but it is important that the appliance be capable of using and producing whichever method is working and pick the method that is working so that it is able to send a message to the appliance registry by encoding the boot status information inside of a message that is legal in any one of these protocols.

Detailed Description Text (57):
Appliance Ownership (Boot Registry)

Detailed Description Text (58):
One objective of the appliance is the ability to boot in a manner controlled by its owner, without the availability of a local boot server. The appliance may be owned by a distant organization that has no shop or facilities in the immediate physical vicinity of the appliance. For example, the appliance could be owned by a service organization such as a news network, newspaper, or insurance company that is very interested in selling insurance, news, or financial services to the personnel in an office environment, and so the owner will install a server attached to the local network of the office environment that is capable of transmitting videos or otherwise to interact with the personnel in the office's local area network environment. This arrangement is analogous to installing a satellite dish to receive network videos and then send them into a small cable network inside the building to be received by a television. Similarly, office personnel have personal computers on their desktops, and instead of installing a satellite disk, a server has been installed that is owned and operated by a video company, network, a newspaper, or the like but that in attached to the office's local area network. The video company, network, or newspaper doesn't wish to send a person out to the office to deal with the appliance, and the office itself doesn't want to have to deal with the appliance. So its very important that the appliance, which is owned and operated by a far-away entity, be controlled by that far-away entity in spite of the fact that the appliance is isolated somewhere within the office. That's why the appliance does not depend on the presence of a boot server configured by the office's information systems department. Instead, the appliance observes some traffic on the local area network (step 108, FIG. 4), computes a set of candidate network parameters (step 110), tests them for validity (step 112), and then, having tested them for validity, transmits messages across the internet to the appliance registry (step 114).

Detailed Description Text (61):
Thus, the appliance sends to the appliance registry a description of its network configuration based on configuration information received from a boot server or based on what the appliance chose to use as its temporary networking configuration by observing the local network. The appliance also sends to the appliance registry an indication of whether it has successfully communicated with any boot server in the local network environment as well.

Detailed Description Text (63):
Another key part of the design is that the ownership table 32 of database 30 of registry 28 (FIG. 3) lists the appliances by identification code and owner of the appliance. Thus, if the company that owns a certain appliance ships it to a certain office, the owner would not need to install anything special in that appliance. Rather, someone at the office simply plugs the appliance into local area network 14 and turns it on. Appliance 18 orients itself over local area network 14, figures out what parameters to use, tests them for validity whether local area network 14 has a boot server or not, and then sends, using one of a variety of protocols (whichever one it can get to work) messages to appliance registry 28 identifying itself by identification number, identifying which version of software the appliance is operating, and indicating whether the appliance needs help in booting. Appliance registry 28 is then able to respond to that message with the identity of the owner, using ownership table 32, which lists owners of appliance as well as other information that would need to be given to appliance 18.

Detailed Description Text (64):
Configuration table 36 of database 30 is specific to the appliance owner, who may store therein a package of configuration records because it is listed as the owner of a given

appliance in ownership table 32. The operator of the registry service lists appliance 18 as belonging to a given owner, in ownership table 32, when the operator of the registry service sells appliance 18 to its owner. As a consequence, the owner can supply configuration information for entry into configuration table 36. When appliance registry 28 receives a boot status message for appliance 28 (which message includes the identification number of the appliance), it records the contents of the boot status message in boot status table 34 and it replies to appliance 18 with the information that the appliance belongs to a certain owner. Appliance 18 now knows the identity of its owner. Appliance registry 28 also replies to appliance 18 with the boot configuration information that the owner may have already loaded in configuration table 36 of database 30. If the owner has not already loaded that information, appliance registry 28 can supply appliance 18 with default configuration information or it can be given a message indicating that the owner has not yet supplied configuration information and that the appliance should simply retry later.

Detailed Description Text (65):
With reference to FIG. 5, the ownership and configuration tables of the registry database are represented together as a single table 60 having a set of columns that include an identification 62 of an appliance, the owner 64 of the appliance, the type 66 of the appliance, configuration attributes 68 to be given to the appliance, and status information 70 received from the appliance in its boot status report.

Detailed Description Text (66):
The recording of boot information in status table 34 of database 30 is important because it enables personnel of the owner to access the database, which they may do manually or automatically, to learn the status of all the appliances that the owner owns and operates. This feature is also important because it is necessary for anyone located anywhere in the world, but also potentially inside the office at which appliance 18 is located, to be able to verify whether the appliance is working. The registry stores all the information it gets from the appliance in the database, including the information that the appliance observed about its network, so that information can be used by other algorithms.

Detailed Description Text (97):
The global service knows where the browser is located (because the request messages that come from the browser include network addressing information), and so it knows where to send the configuration information for the appliance (step 210). The appliance is not really a functioning part of the system yet, because the appliance has not been properly configured, but the appliance has had success in transmitting its status message. Thus, the appliance is known to the system. Because the appliance is known, when the network service is accessed by the web browser, the network service can usually recognize that an appliance at the same local network ought to be the proper serving appliance. Recall that when the boot status message is sent from an appliance to a registry, there is information included in the status message such as observed network information and the appliance identity, which is stored in the registry database. Accordingly, when a network service provider receives a request from a given browser it can tell from the return addressing information on the browser's request that the browser is almost certainly from the same local network environment as the particular appliance, because return addressing information falls within the range of IP addresses that the appliance reported as the set of IP addresses that were being used in its local area network. In this case the network service can communicate configuration information to the appliance by using steps 224 and 226 as described above.

Detailed Description Text (98):
Isolated Boot

Detailed Description Text (99):
Moreover, certain networking environments have a very secure network configuration in which it is not possible for the appliance to communicate with the external public Internet (step 212). It might be impossible to communicate with the public Internet from the appliance, or the network may have been configured so that a particular computer can't communicate to the public Internet without being given some sort of code or password or extra piece of information. This could be intentionally designed to make it very difficult for the appliance to communicate with the Internet without being authorized or it could be an accident of the way the network was configured. In order to permit booting and configuration of the appliance in such environments, the appliance would display on its LCD panel some information (step 214) that a user could enter into the user's web browser on one of the computers on the local area network, in

order to enable the user to communicate from the web browser to the appliance. The user must be present at the appliance in order to see the LCD panel. The user enters a code on the user's local computer in order to access the appliance.

Detailed Description Text (101):
When a user first receives an appliance and plugs it into the wall, the user might also receive a card that, in addition to instructing the user to plug the appliance into the wall, instructs the user to perform a simple test to allow the user to know whether the appliance was plugged into the wall correctly. For example, the user could plug the appliance into the wall, and a little LED flashes green on the appliance box if the appliance is operating correctly, but an LED might either flash red to indicate that the appliance has not been installed correctly or yellow to indicate that something minor is wrong. The card provided with the appliance can direct the user to a troubleshooting page on the World Wide Web, which is a page provided by the appliance registry, where the user would be shown a troubleshooting page that the user can use to figure out what is wrong with the appliance and why it wasn't booting or working correctly in the user's network environment. The troubleshooting page, in order to provide the user with proper assistance, can look into the registry database to see whether the appliance has reported a status into the status table and what the status as reported in the table is.

Detailed Description Text (105):
If none of the various techniques for booting and configuring the appliance succeed, an authorized person may call a 1-800 to obtain the required information to boot and configure the appliance.

Detailed Description Text (106):
We now set forth the above-described booting algorithm ("sneaky boot") in more detail.

Detailed Description Text (107):
Remote Boot; Detailed Description

Detailed Description Text (108):
Sneaky boot assumes an Internet appliance attached to a LAN that has at least some HTTP connectivity and possibly some IP and DNS connectivity with the public internet. It also assumes it doesn't have to deal with protocols such as Appletalk, WINS, SOCKS, IPX, NetBeui, etc.

Detailed Description Text (109):
We list below the parameters involved in sneaky boot:

Detailed Description Text (124):
The sneaky boot algorithm initializes the values of the parameters to any recommended or last-known-useful values. It sets the associated confidence level for each parameter based on the last known working time for that parameter. Only the mac_addr, serial_number, and registry parameters will be set with confidence level infinite.

Detailed Description Text (125):
The sneaky boot algorithm is basically a search algorithm. It tries a remote procedure call to the appliance registry (failing quickly if required parameters are not high confidence). If successful, the boot algorithm configures the appliance with the received network configuration. If there is a failure, the algorithm picks the parameter that has not been verified with high confidence in the longest time and runs its improvement procedures. The algorithm continues until it is successful.

Detailed Description Text (179):
In all the cases in which sneaky boot fails, the user contacts the appliance registry web site from the user's Web browser. The site contains a number of trouble-shooting pages, some of which will try to get the configuration information through the user's web browser to the appliance.

Detailed Description Text (181):
If IPv6 is supported everywhere, the sneaky boot algorithm will be relatively simple.

Detailed Description Text (188):
With the technology trends taking place today, it is possible to make the cost of owning or operating the appliance be negligible. In particular, the most an owner would typically want to be able to do is to find out how the appliance is performing and perhaps tell it to stop performing some function or to cease operations. An owner might

want to be able to change the configuration somewhat, but not to travel to the user's office to modify the appliance. For example, the cost of replacing the appliance's disk drive would include the cost of the drive and the cost of having someone stop by the user's office and open up the appliance case, which together would typically exceed the cost of a replacement appliance.

Detailed Description Text (208):
PASSIVE: the records are not actively copied between SODA boxes via the usual distributed database mechanisms; instead these records are managed specially by the booting or routing logic, for example;

Detailed Description Text (211):
As part of remote boot, the appliance uses the factory-installed data and the MAC address or other information to contact the appliance registry and obtain the "OWNER" records for the appliance. The registry holds some records such as:

Detailed Description Text (222):
Because the sneaky boot algorithms tell all modules that the local appliance is named "Coke.box.mtv.sn," the database manager module and the routing module look for MAPS entries and see the single MAPS entry with method FETCH above.

Detailed Description Text (242):
Booter 48 is responsible for configuring the appliance. It runs the remote boot algorithm, collects database records, and then launches the other modules.

Detailed Description Text (265):
One of the main challenges in implementing a SODA appliance is to make it behave as a true appliance. We discuss booting and the power switch.

Detailed Description Text (266):
Booting

Detailed Description Text (268):
After an appliance boots, booter 48 runs the sneaky boot algorithm in order to obtain enough of an IP dialtone to connect to the appliance registry. The appliance registry is master inventory of all of the appliances sold. The appliance registry hands booter 48 the appropriate network configuration information to obtain a continuous IP dialtone and connect to its roots.

Detailed Description Text (271):
1) the GUID is not present in the appliance registry. The appliance will not boot and the quick-start manual tells the user of the appliance to visit the troubleshoot page at the appliance registry site. The troubleshoot page asks the user to verify the GUID. If the GUID is correct and not present, the user is asked to contact the corporate headquarters for the supplier of the appliances.

Detailed Description Text (273):
3) The network configuration is not present. The appliance will not boot and the quick-start manual tells the user of the appliance to visit the troubleshoot page at the appliance registry site. The troubleshoot page tells the user to verify whether the appliance is appropriately connected and to submit the state of the appliance (the GUID and what LEDs are flashing). The registry responds with a page that redirects the user to the user's MIS department with a diagnosis of the problem.

Detailed Description Text (275):
As an additional trick, once the user connects with a Web browser to the appliance registry, the boot registry can put the network configuration information through the user's web browser into the appliance.

Detailed Description Paragraph Table (1):
APPENDIX A TO SELF-ORGANIZING DISTRIBUTED APPLIANCES # # Copyright (c) 1998-1999 by SightPath, Inc., Waltham, Massachusetts. # # # sampledb.mx # # Draft of registry records. # # This is for doing a booter test. # # We will make seven-up boot by talking to `registry.sightpath.net`. # `registry.sightpath.net` will really be seven-up:80 for testing? # There will only be a few known boxes # # OWNERSHIP # Registry tracks box ownership using traditional deed system. # # Boxes are named in this table using absolute (owner-independent) # names. # # Deeds are used to provide an audit trail for SightPath support # purposes. # (unigname.box.registry.sn # vbox.mac-addr OWNERSHIP [DEED nnnnn]) # number of latest deed (1.01:02:03:0a:0b:0c.box registry.sn OWNERSHIP

{DEED 1.deed.registry.sn}) (2.01:02:03:0a:0b:0c.box.registry.sn OWNERSHIP {DEED 2.deed.registry.sn}) (3.01:02:03:0a:0b:0c.box.registry.sn OWNERSHIP {DEED 3.deed.registry.sn}) # DEED # Registry contains deeds. # (nnnnn.deed.registry.sn DEED {OLD-OWNER whoever.sn # grantor NEW-OWNER whoever.sn # grantee CHASSIS ccc # id info for box MAC-ADDR mmm # id info for box TRANSFER-TIME t # when ownership changed PREVIOUS-DEED dd # grantor's earlier deed FACTORY-ISSUE true/false # indicates original deed SIGNED ss}) # proof of validity (1.deed.registry.sn DEED {OLD-OWNER sightpath NEW-OWNER otoole.user.sn. CHASSIS hunh MAC-ADDR 1.01:02:03:0a:0b:0c TRANSFER-TIME 944 PREVIOUS-DEED none FACTORY-ISSUE true SIGNED fix.me}) (2.deed.registry.sn DEED {OLD-OWNER sightpath NEW-OWNER clearmedia.publisher.sn CHASSIS what MAC-ADDR 2.01:02:03:0a:0b:0c TRANSFER-TIME 1944 PREVIOUS-DEED none FACTORY-ISSUE true SIGNED fix.me}) (3.deed.registry.sn DEED (OLD-OWNER sightpath NEW-OWNER clearmedia.publisher.sn CHASSIS whazzat MAC-ADDR 3.01:02:03:0a:0b:0c TRANSFER-TIME 1944 PREVIOUS-DEED none FACTORY-ISSUE true SIGNED [ix.me)) # # BOOT-STATUS # As each box boots, it "check in" at the registry. The registry keeps # track of these boxes via these BOOT-STATUS records. The owner's # config root box copies these BOOT-STATUS records (via the Mapper) so # that the box status can be shown to the administrator via the Admin # GUI. # # Boxes are named in this table using owner-specific moniker suffixes. # (uniqname.box.owner.sn # vbox.mac-addr BOOT-STATUS (REPORT-TIME t # R's time when box called in APPARENT-IP ipaddr # R observed this ip src addr REPORT-METHOD http.vertline.dns.vertline.udp # how box called into R GAVE-BOOT-CONFIG-STAMPED t # what R gave to box, maybe USING-BOX-INFO mxccc # box's BOX-INFO in usc Box-INFO-SOURCE registry.vertline.dhcp.vertline.browser.vertline.owner # from box OTHER-INFO-FROM-BOX mxrrr # from box VERSION vv # from box BOOT-COUNT nn # from box SW-RESET-COUNT gg # from box UPTIME secs # from box LAST-REGISTRY-CONTACT cc # from box REPORTED-OWNER whoever.sn # from box LAST-OWNER-CONTACT secsago}) # from box (1.01:02:03:0a:0b:0c.box.clearmedia.sn BOOT-STATUS {REPORT-TIME 8000 APPARENT-IP 208.246.45.www REPORT-METHOD http GAVE-BOOT-CONFIG-STAMPED 5000 USING-BOX-INFO (1.01:02:03:0a:0b:0c.box.clearmedia.sn BOX-INFO {USE-DHCP 1}) BOX-INFO-SOURCE registry OTHER-INFO-FROM-BOX none VERSION 1 BOOT-COUNT 5 SW-RESET-COUNT 1 UPTIME 500 LAST-REGISTRY-CONTACT 500 REPORTED-OWNER registry.sn LAST-OWNER-CONTACT 0}) # # BOOT-CONFIG # Registry holds config data for boxes to give them help. # Boxes are named in this table using owner-specific moniker suffixes. # (uniqname.box.owner.sn # vbox.mac-addr BOOT-CONFIC {MOD-TIME t # R's time when owner wrote this ASSIGNED-CONFIG-ROOT whoever.sn # from owner BOOT-PACKAGE {LIB-RECORDS mxrrr BOOT-RECORDS mxrrr ONLY-VERSION n TIMESTAMP t}}) (1.01:02:03:0a:0b:0c.box.clearmedia.sn BOOT-CONFIG (MOD-TIME 1000 ASSIGNED-CONFIG-ROOT registry.sn BOOT-PACKAGE (LIB-RECORDS {(1.01:02:03.0a:0b:0c.box.clearmedia.sn BOX-INFO {USE-DHCP 1})) BOOT-RECORDS {BOX-NAME 1.01:02:03:0a:0b:0c.box.clearmedia.sn} ONLY-VERSION 1 TIMESTAMP 915725062}}) (2.01:02:03:0a:0b:0c.box.clearmedia.sn BOOT-CONFIG {MOD-TIME 5000 ASSIGNED-CONFIG-ROOT clearmedia.sn BOOT-PACKAGE {LIB-RECORDS ((2.01:02:03:0a:0b:0c.box.clearmedia.sn BOX-INFO {USE-DHCP 1}) (clearmedia.sn PROVIDER-INFO {ROOT 1.01:02:03:0a:0b:0c.box.clearmedia.sn}) (1.01:02:03:0a:0b:0c.box.clearmedia.sn ADDRESS-INFO (IP 80.123.45.19 PORT 80}) (2.01:02:03:0a:0c:0c.box.clearmedia.sn MAPS {{SUFFIX 2.01:02:03:0a:0b:0c:box.clearmedia.sn TABLE MAPS PROVIDER clearmedia.sn HOW PREFETCH}})) BOOT-RECORDS {BOX-NAME 2.01:02:03:0a:0b:0c.box.cleamedia.sn} ONLY-VERSION 1 TIMESTAMP 915725062}}) # # ONLY-VERSTON is to specify that these records should be used only if the # sonoma box is running this version of the sonoma software. # # # BOOT-PACKAGE # Registry receives boot packages from box owners. # Registry daemon (Reggie) checks these packages and moves them into # the BOOT-CONFIG table. # (uniqname.box.owner.sn # vbox.mac-addr BOOT-PACKAGE (RECORDS mxrrr ONLY-VERSION 1 TIMESTAMP L}) (2.01:02:03:0a:0b:0c.box.clearmedia.sn BOOT-PACKAGE {RECORDS ((2.01:02:03:0a:0b:0c.box.clearmedia.sn BOX-INFO {USE-DHCP 1})) ONLY-VERSION 1 TIMESTAMP 91575062}) # # PRINCIPAL # Registry is pretending to be secure by tracking owners with keys, # account names, etc. # # These entries are for support to track the people who really are # supposed to control the boxes. # # The SONOMA-SUFFIX entry is important because it defines the part # of the Sonoma Network namespace that will be used for # boxes under this ownership. # (whoever.sn PRINCIPAL {EMAIL-CONTACT ee # user@domain USERNAME u # username for secure login PASSWORD p # password for secure login SONOMA-SUFFIX owner.sn # suffix used for owner-specific mks SONOMA-HOSTNAME h # name of Sonoma published host SONOMA-PORT nn # port to contact Sonoma PUBLIC-KEY pk # key for secure signatures SHARED-KEY sekrit}) # key for okay signatures (otoole.user.sn PRINCIPAL (EMAIL-CONTACT otoole@clearmedia.com USERNAME otoole PASSWORD Bcareful SONOMA-SUFFIX clearmedia.sn SONOMA-HOSTNAME mypublisher.clearmeida.com SONOMA-PORT 8082 PUBLIC-KEY pk SHARED-KEY sekrit}) (clearmedia.publisher.sn PRINCIPAL (EMAIL-CONTACT publishing@clearmedia.com USERNAME otoole PASSWORD Bcareful SONOMA-SUFFIX clearmedia.sn SONOMA-HOSTNAME mypublisher.clearmedia.com SONOMA-PORT 8082 PUBLIC-KEY pk SHARED-KEY sekrit}) # # The rest of these records are to configure the librarian and/or # to keep any running Sonoma servers on Registry happy and quiet. # # # MAPS # Registry's

librarian does not listen to any other librarian (it does # not have any MAPS records referencing other boxes). It does "accept" # BOOT-PACKAGE submissions (the config root boxes have MAPS/RPORT # records to food the BOOT-PACKACE records to the registry). #

Detailed Description Paragraph Table (2):
(1.01:02:03:0a:0b:0c.box.clearmedia.sn MAPS ({TABLE BOOT-PACKAGE PROVIDER registry.sn SUFFIX " " HOW REPORT} # (along with all the other usual MAPS record elements))) (the-registry.registry.sn MAPS ({TABLE MAPS PROVIDER registry.sn SUFFIX sn HOW PASSIVE} {TABLE OWNERSHIP PROVIDER registry.sn SUFFIX registry.sn HOW PASSIVE} {TABLE DEED PROVIDER registry.sn SUFFIX registry.sn HOW PASSIVE} (TABLE BOOT-STATUS PROVIDER registry.sn SUFFIX sn HOW PASSIVE} {TABLE BOOT-CONFIG PROVIDER registry.sn SUFFIX sn HOW PASSIVE} (TABLE BOOT-PACKACE PROVIDER registry.sn SUFFIX sn {TABLE PRINCIPAL PROVIDER registry.sn SUFFIX registry.sn HOW PASSIVE})) # # PROVIDER-INFO # For completeness, we tell the registry about itself? # (registry.sn PROVIDER-INFO {ROOT the-registry.registry.sn)) (the-registry.registry.sn ADDRES-INFO {IP 208.246.45.nnn PORT 80 HOSTNAME registry.sightpath.net}) (the-registry.registry.sn ROUTER-INFO {PREFERRED-ZONE (0.0.0.0/32) REGULAR-ZONE (0.0.0.0/32)}) (the-registry.registry.sn BOX-INFO {USE-DHCP 1}) # # How much of this do we need to keep Sonoma happy for now? # # # - At the RUI (Registry User Interface), administrator add a BOOT-STATUS # record to indicate there is a new box. The BOOT-STATUS record initially # indicates that the box has never been heard from. # # Possible complication: someone powers-on a box before the BOOT-STATUS # record has been created. # # - [xxx - someone] creates a MAPS/BOOT-PACKAGE/REPORT record element on the # config root boxes. (Perhaps this happens as part of the normal procedure # or configuring a sonoma roor box.) Similarly there's a # MAPS/BOOT-STATUS/PREFETCH element. # # - Admin GUI uses the BOOT-STATUS records when building the page that # displays the status of each box. # # - Admin GUI allows the administrator to modify the BOOT-INFO record fields. # As a side-effect of modifying a box's BOOT-INFO, the Admin GUI creates a # BOOT-PACKAGE record for that box. The Admin GUI creates the BOOT-PACKAGE # records each time the information within changes (more or less). The # Admin GUI should not expect to be able to readback the BOOT-PACKAGE # records (the Mapper will be shunting them off to the registry). # # - As BOOT-PACKAGE records arrive at the registry, the Reggie daemon/server # extracts them from the library, deletes them, and inserts the information # into box's BOOT-CONFIG record as the BOOT-PACKAGE field. At the same # time, Reggie sets the BOOT-CONFIG record's MOD-TIME field; this value is a # POSIX time value intended to facilitate troubleshooting. # # - As an arbitrary appliance box boots, it uses a cgi-bin program (reg) to # contact the Registry. This "checkin" has two functions: to fetch the # BOOT-PACKAGE field from the box's BOOT CONFIG record, and to update the # box's BOOT-STATUS record. # # The box provides two pieces of information: its (virtual) MAC address # (e.g., "1.01:02:03:0a:0b:0c"), and an Mx hash. USing the MAC addrsss, reg # constructs a box moniker (e.g., "1.01:02:03:0a:0b:0c.box.clearmedia.sn") # via the following steps: # - Append ".box.registry.sn" to the MAC address, giving an owncership # moniker like "1.01:02:03:0a:0b:0c.box.registry.sn". # - Fetch that OWNERSHIP record, and get the DEED field. The value of this # DEED field is a deed moniker, like "3.deed.registry.sn". # - Fetch that DEED record, and get the NEW-OWNER field. This field is a # prinicipal moniker. # - Fetch that PRINCIPAL record, and get the SONOMA-SUFFIX field. This # field is the suffix used for all moniker of this provider (e.g., # "clearmedia.sn"). The box moniker is constructured by concatenating the # MAC address with ".box." and with the suffix, giving something like # "1.01:02:03:0a:0b:0c.box.clearmedia.sn". # # The hash contains status information about the box, as determined by the # box's booter scripts. This status information is the "from box" fields of # the BOOT-STATUS examples, above. Reg adds a few fields to the hash (the # non-"from box" fields). # # Reg writes a BOOT-STATUS record into the library. The box moniker is the # key, and the hash is the value. # # Reg uses the box moniker fetch a BOOT-CONFIG record. Reg extracts the # BOOT PACKAGE field from that record. Reg returns the box moniker and the # BOOT-PACKAGE field to the box. # # The booter scripts dump the LIB-RECORDS into library.mx, and dump the # BOOT-RECORDS into boot.mx. # # The booter scripts also construct an ADDRESS-INFO record for the box using # the BOX-INFO information. The booter scripts might need to perform. DHCP # operations to determine what IP field to use. The booter scripts put the # ADDRESS-INFO record into library.mx. # # The booter then starts the Sonoma servers. #

CLAIMS:

1. An system for booting an appliance, comprising:

appliance; and

a local network of computers;

the appliance being programmed to transmit a message over the local network to obtain a source of booting parameters; and if the appliance receives the message containing booting parameters, to test the parameters by attempting to send and receive network messages over the local network; and if the appliance fails to receive a message containing acceptable parameters, to observe the local network passively, to develop candidate parameters based on network traffic and network addresses in use on the network, and to test the candidate parameters by sending and receiving packets in the local network.

5. The system of claim 1 further comprising an appliance registry, wherein the appliance is programmed to transmit, after the step of testing the candidate parameters, a boot status message to the appliance registry while the appliance registry is located outside the local network.

6. The system of claim 5 wherein the appliance is programmed to select a temporary address for the appliance and to use the temporary address to transmit the boot status message to the appliance registry, if the appliance fails to receive a message containing booting parameters.

12. The system of claim 1 wherein the appliance is programmed to transmit the message to obtain the source of booting parameters by transmitting a bootp, DHCP, or rarp request.

13. The system of claim 1 wherein the booting parameters comprise an IP address of the appliance.

14. The system of claim 1 wherein the booting parameters comprises a subnet mask of the appliance.

15. The system of claim 1 wherein the booting parameters comprise an IP addresses of at least one router.

16. The system of claim 1 wherein the booting parameters comprises identification of at least one name server.

18. A system for obtaining configuration information for an appliance, comprising:

an appliance having a unique global identifier; and

an appliance registry;

the appliance being programmed to transmit a boot status message from the appliance using its global identifier, in a local network, to an appliance registry located outside of the local network;

the appliance registry being programmed to store boot status information from the boot status message in a database at the appliance registry, the boot status information including information pertaining to the local network as observed by the appliance; and

the appliance registry being programmed to transmit configuration information from the appliance registry to the appliance based on the global identifier of the appliance.

19. The system of claim 18 wherein the boot status information comprises identity of the appliance.

22. The system of claim 18 wherein the appliance is programmed to observe the local network to develop candidate parameters based on network traffic and network addresses in use, the appliance using the candidate parameters for communicating with the appliance registry to transmit the boot status message.

23. A system for obtaining configuration information for an appliance, comprising:

an appliance; and

an appliance registry;

the appliance being programmed to transmit a boot status message from the appliance in

a local network to an appliance registry located outside of the local network using a plurality of redundant transmission protocols; and

the appliance registry being programmed to transmit configuration information from the appliance registry to the appliance using a plurality of redundant transmission protocols.

28. The system of claim 23 wherein the plurality of protocols comprises encoding the boot status message into a universal resource locator and sending an HTTP request comprising the universal resource locator from the appliance to the appliance registry.

29. The system of claim 23 wherein the plurality of protocols comprises encoding the boot status message as an imaginary host name and sending a DNS message from the appliance to the appliance registry as a request to obtain information pertaining to the imaginary host name.

30. The system of claim 23 wherein the plurality of protocols comprises sending the boot status message from the appliance to the appliance registry as an e-mail message.

39. A system for obtaining configuration information for an appliance, comprising:

an appliance;

an appliance registry; and

a registry database;

the appliance registry being programmed to store ownership information for the appliance in the registry database

the appliance being programmed to transmit a boot status message from the appliance to the appliance registry while the appliance registry is located outside of a local network of the appliance, the boot status message including information pertaining to the local network as observed by the appliance;

the appliance registry being programmed to transmit configuration information from the appliance registry to the appliance in response to the boot status message.

48. The system of claim 39 wherein the appliance is programmed to observe the local network to develop candidate parameters based on network traffic and network addresses in use, the appliance using the candidate parameters for communicating with the appliance registry to transmit the boot status message.

49. A method of obtaining configuration information for an appliance located in a local network, comprising the steps of:

transmitting a boot status message from the appliance to an appliance registry located outside of the local network using a plurality of redundant transmission protocols; and

transmitting configuration information from the appliance registry to the appliance using a plurality of redundant transmission protocols.

50. The method of claim 49 further comprising the steps of:

observing the local network to develop candidate parameters based on network traffic and network addresses in use; and

using the candidate parameters for communicating with the appliance registry for transmission of the boot status message.

51. A method of obtaining configuration information for an appliance located in a local network, comprising the steps of:

transmitting a boot status message from the appliance to an appliance registry located outside of the local network, the boot status message including information pertaining to the local network as observed by the appliance;

storing the boot status information from the boot status message in a database at the appliance registry; and

transmitting configuration information from the appliance registry to the appliance.

52. The method of claim 51 further comprising the steps of:

observing the local network to develop candidate parameters based on network traffic and network addresses in use; and

using the candidate parameters for communicating with the appliance registry for transmission of the boot status message.

53. A method of obtaining configuration information for an appliance located in a local network, comprising the steps of:

storing ownership information for the appliance at a registry database of an appliance registry located outside of the local network;

transmitting a boot status message from the appliance to the appliance registry, the boot status message including information pertaining to the local network as observed by the appliance;

transmitting configuration information from the appliance registry to the appliance in response to the boot status message.

54. The method of claim 53 further comprising the steps of:

observing the local network to develop candidate parameters based on network traffic and network addresses in use; and

using the candidate parameters for communicating with the appliance registry for transmission of the boot status message.

55. A method of booting an appliance in a local network, comprising the steps of:

transmitting a message over the local network to obtain a source of booting parameters;

attempting to receive, in response to the message transmitted by the appliance, a message that contains booting parameters for the appliance;

if the appliance receives the message containing booting parameters, testing the parameters by attempting to send and receive network messages over the local network; and

if the appliance fails to receive a message containing acceptable parameters, observing the local network passively, developing candidate parameters based on network traffic and network addresses in use on the network, and testing the candidate parameters by sending and receiving packets in the local network.

57. An appliance programmed to transmit a message over a local area network to obtain a source of booting parameters, and being programmed to determine if the appliance receives a message containing booting parameters, to test the parameters by attempting to send and receive network message over the local network; and if the appliance fails to receive a message containing acceptable parameters, to observe the local network passively, to develop candidate parameters based on network traffic and network addresses in use on the network, and to test the candidate parameters by sending and receive packets in the local network.

L12: Entry 27 of 50 File: USPT Dec 26, 2000

DOCUMENT-IDENTIFIER: US 6167567 A
TITLE: Technique for automatically updating software stored on a client computer in a networked client-server environment

Application Filing Date (1):
19980505

Detailed Description Text (6):
Thusfar described, to permit conventional network-based updating of firmware 45 for terminal adapter 40, the manufacturer will place requisite update files either on its FTP site and/or its web site and permit its customers to access, through their client PCs, either site and download the necessary files. To do so, a user stationed at client PC 10 will typically establish either an FTP connection through network 50 to server 70, specifically to FTP server 82, or an HTTP connection to HTTP server 85. Having done so, the user will then navigate through the FTP site or web site to locate and download the proper update files. Thereafter, these files will then install the update either off-line, through user initiated-execution of typically an appropriate "setup" (setup.exe) file, or through automatic initiation through the web site. Once the update has completed, the client PC will then terminate the FTP or HTTP connection to server 70.

Detailed Description Text (20):
As shown, client PC 10 comprises input interfaces (I/F) 210, processor 220, memory 230, communication interfaces 250, and output interfaces 260, all conventionally interconnected by bus 270. Memory 230, which generally includes different modalities, includes illustratively random access memory (RAM) 235 for temporary data and instruction store, diskette drive(s) (not specifically shown) for exchanging information, as per user command, with floppy diskettes, and a non-volatile store typically implemented by hard disk drive(s) 232 which are generally magnetic in nature. Terminal adapter 40 is connected through leads 35 to communication interfaces 250, e.g., implementing an RS-232 serial port, and specifically contains firmware 45 that is to be updated.

Detailed Description Text (23):
Memory 230, specifically hard disk 232, stores application programs 240, including our inventive client update application 15, and operating system 245. Application 15 includes the software updating application (application 500 discussed in detail below in conjunction with FIGS. 5A-5D), the configuration process and appropriate software modules to perform a product de-registration process (process 1300 discussed in detail below in conjunction with FIG. 13). Operating system 245, which is illustratively the Microsoft Windows 95, Windows 98 or Microsoft Windows NT 4.0 or 5.0 operating system, includes registry 247 and component object model (COM) 249. Inasmuch as the registry and COM are well-known in the art, we will only discuss below those aspects of these software modules that are specifically germane to the present invention.

Detailed Description Text (28):
Configuration application 25 provides a graphical user interface (GUI) through which a user stationed at client PC 10 can interact with and configure application 500. The extent of this configuration, which will become evident from the screen displays shown in FIGS. 14A-14D, includes, e.g., detecting and registering any software product for use with application 500; enabling event logging of update activity; and for each software product so registered, e.g.: scheduling update intervals for that product, specifying a network connection method and connection parameters for connecting to a network server for updating that product, and confirming all updates of that product with a user. Configuration application 25 stores and accesses, as symbolized by line 316, configuration information within local storage 310 which is collectively implemented by requisite storage space within hard disk 232 (see FIG. 2). Local store

310 includes, as shown in FIG. 3, O/S registry 247. Additionally, configuration application 25 also communicates, as symbolized by line 325 and through COM 320, with updating application 500.

Detailed Description Text (37):
If a custom web site update is to occur, then client PC 10 performs operations 458. Through these operations, the client PC obtains, from registry 247, the URL of the update web site for product i, initiates execution of client web browser 370 (see FIG. 3) and passes that URL to the browser for retrieving an "INDEX.HTM" file. The browser then establishes, as symbolized by line 460 in FIGS. 4A and 4B, an HTTP connection with the update web site, here illustratively HTTP server 85, and opens the INDEX.HTM page at this site. Thereafter, the user then interacts, through the client PC and the browser, with the update web site to download appropriate update files for product i and install the update files accordingly. Once the browser is closed and execution transferred back to application 500, operations 462 occur to ask the user whether (s)he completed the update. If the user indicates that the update was completed, then operations 464 occur to update the version number of product i, stored in the registry, to that specified in the update and schedules the next update for this product. Next, through operations 465, if the update script specifies that the user is to re-boot the client PC in order to complete the update, then a suitable notification is displayed to the user instructing the user to do so. Execution of application 500 is then finished for product i. Alternatively, should the user indicate,as a result of operations 462, that the update was not completed, then operations 465 occur to appropriately inform the user about re-booting, if required. In this case, execution of application 500 then finishes, via exit point 466, without any change being made to the product i version number stored in the registry.

Detailed Description Text (39):
Once all the "run" files have fully executed, operations 484 are performed to determine whether any user interaction is required, such as issuing an notification to the user and requesting confirmation back from the user. If such interaction is required, then operations 486 occur to determine whether the update has completed. If the user provided such confirmation, then operations 488 occur to update the version number, stored in the registry, of product i and to schedule the next update of this product. These operations are performed, after operations 484, if no user interaction is required. Next, through operations 492, if the update script specifies that the user is to re-boot the client PC in order to complete the update, then a suitable notification is displayed to the user to do so. Once this occurs, execution of application 500, at client PC 10, is finished for product i. Alternatively, if the user signifies that the update was not completed, then operations 488 are not performed and then operations 492 occur to appropriately inform the user about re-booting, if required. In this case, the version number is not updated and the next update of product i is not scheduled.

Detailed Description Text (47):
If the update is to proceed through the update script, then decision block 534 routes execution, via NO path 536, to block 537. This latter block, when executed, detects the specific O/S then operating in the client PC and processes those sections of the script which are: (a) O/S-independent, and (b) correspond to the detected O/S. This processing entails, given the parameters (as discussed below) in these sections, establishing appropriate lists of "copy" and "run" files to download and determining source directories on the FTP server at which these files are located and destination directories on the client PC into which these files are to be copied and, for the "run" files, executed. The file names for these "copy" and "run" files collectively define an "update file name" set with the corresponding update files themselves defining an "update file" set. Thereafter, execution proceeds to block 538. This block, when executed, downloads each of these "copy" files from its source directory on the FTP site into its destination directory on the client PC. Once this occurs, execution proceeds to decision block 539 which tests whether all the "copy" files were successfully downloaded. If the download of all the "copy" files did not succeed, e.g., a "copy" file could not be downloaded or an error arose during its downloading, then decision block 539 routes execution, via NO path 541, to block 542. If the FTP connection still exists to the update FTP site, block 542 simply terminates this connection. Thereafter, execution is directed, via path 532, to block 588 and so forth, as described above, after which execution ultimately exits from application 500. Alternatively, if all the "copy" files were successfully downloaded, then decision block 539 routes execution, via YES path 540, to block 543. This latter block, when executed, downloads each of the listed "run" files from its source directory on the FTP site into its destination directory on the client PC and then executes each of these files in the order downloaded. Once this occurs, execution proceeds to decision block

545 which tests whether all the "run" files were successfully processed, i.e., downloaded and executed. If the processing of these files did not succeed, e.g., a "run" file could not be downloaded or executed or an error arose during its downloading or execution, then decision block 545 routes execution, via NO path 546, to block 547. If the FTP connection still exists to the update FTP site, this latter block simply terminates this connection. Thereafter, execution is directed, via path 532, to block 588 and so forth, as described above, after which execution ultimately exits from application 500. Alternatively, if all the "run" files successfully processed, then decision block 545 routes execution, via YES path 548, to block 549. To the extent any "run" file is still executing, block 549 simply waits for the last such file to complete its execution. Once all such files have fully executed to install the update, then block 550 executes. This block, when executed, automatically schedules a date for the next update for product i. Thereafter, execution proceeds to decision block 552 which determines, based on the update confirmation configuration setting, whether the user required that (s)he be notified of each update and enter his(her) subsequent acknowledgement of its completion. If so, decision block 552 routes execution, via YES path 554, to block 556. This latter block, when executed, prompts the user to confirm, typically through a mouse click on a button then appearing on the display (display 283 in FIG. 2), that the update has completed. If the user then confirms that the update was completed, then decision block 558, shown in FIGS. 5A and 5B, routes execution, via YES path 560, to block 562. This latter block, when executed, updates the version number, stored in the registry, for product i with the version number provided in the update script. Once this occurs, block 563 determines whether the update script specifies that the user is to re-boot the client PC in order to complete the update. If such a re-boot is required, then this decision block routes execution, via YES path 565, to block 566. This latter block, when executed, displays a suitable notification to the user to re-boot the client PC. Once this occurs, execution proceeds to block 567. Alternatively, is such a re-boot is not required, then execution proceeds directly to block 567 via NO path 564 emanating from decision block 563. Block 567 then executes to terminate the FTP connection. Thereafter, execution proceeds, via path 568, to decision block 582 which, based on a configuration setting, determines whether the user is to confirm updates. If the user has required such confirmation, then decision block 582 routes execution, via YES path 583, to block 585. This latter block, when executed, notifies the user accordingly of the next scheduled update and requests the user to confirm it. Once the user so confirms the update, typically by clicking a displayed "OK" button with his mouse, execution proceeds to block 587 which logs an event, here a successful script-based update. Once the event is logged, execution exits from application 500. Alternatively, if the user has not requested update confirmation, then execution proceeds, via NO path 584 emanating from decision block 582, directly to block 587.

Detailed Description Text (49):
However, once the browser is able to open page INDEX.HTM at the update web site, the user then interacts, through the client PC and the browser, with the update web site to download appropriate update files for product i and install the update files accordingly. As such, execution passes, via YES path 571 emanating from decision block 570, to block 573. Since actual downloading and installation of the update files will proceed through the web site and the browser and with no involvement of application 500 during that interval, block 573 merely waits until the user has closed the browser. Once this occurs, execution proceeds from block 573 to block 574. This latter block, when executed, automatically schedules the next update for product i. Thereafter, decision block 575 executes to determine, based on questioning the user and soliciting an appropriate response, whether the custom web site update completed. If the user indicates, typically through a mouse click on an appropriate button then appearing on the display, that the update completed, decision block 575 routes execution, via YES path 576, to decision block 577. This latter decision block determines whether the update script specifies that the user is to re-boot the client PC in order to complete the update. If such a re-boot is required, then this decision block routes execution, via YES path 579, to block 580. This latter block, when executed, displays a suitable notification to the user to re-boot the client PC. Once this notification occurs, execution proceeds to block 581. Alternatively, if such a re-boot is not required, then execution proceeds to block 581, via NO path 578 emanating from decision block 577. Block 581 updates the version number, stored in the O/S registry, for product i to that specified by the update script. Thereafter, execution proceeds to block 582 and so forth, as described above, after which execution ultimately exits from application 500. Here, the occurrence of a successful web-based update of product i will be logged as an event by block 587. If the custom web site update did not complete, then decision block 575 will route execution, via NO path 597 and path 568, directly to block 582 and so forth, as described above, after which execution ultimately exits from application 500.

Here, the occurrence of an incomplete web-based update of product i will be logged as an event by block 587.

L12: Entry 34 of 50 File: USPT Oct 26, 1999

DOCUMENT-IDENTIFIER: US 5974547 A
TITLE: Technique for reliable network booting of an operating system to a client computer

Abstract Text (1):
A technique, specifically apparatus and accompanying methods, for use in a client-server environment for booting an operating system (O/S), such as a 32-bit personal computer (PC) O/S, on a client computer through a networked connection to a server. Specifically, the server stores an image of a client hard disk including the client O/S and desired applications. During a boot process, a procedure, which is compliant with both an interrupt handler in the client and a network driver kernel in the client O/S, is installed in the client. Based on client O/S resources then available when, during the boot process, the client requests a local hard disk access to a particular sector, the procedure will re-direct that request, to the network file server, through a network driver kernel in the client O/S rather than through a client interrupt handler. Each such request is processed, to provide physical sector read or write access, through my inventive random access trivial file transfer protocol (RATFTP) server executing in the network server. Advantageously, the source of the sectors remains transparent to the client O/S, while it is being booted from a network connection, in lieu of a local hard disk drive. Hence, client hard disk emulation occurs seamlessly and continuously throughout the entire boot process even though, during this process, the client processing mode changes from real to protected and the client O/S resets and gains control of a client network adapter.

Application Filing Date (1):
19980320

Brief Summary Text (3):
The invention relates to a technique, specifically apparatus and accompanying methods, for use in a client-server environment for booting an operating system (O/S) on a client computer through a networked connection to a server. This technique is particularly, though not exclusively, suited for use in booting 32-bit personal computer (PC) operating systems and can be advantageously used to simplify system administration and significantly reduce administrative costs in, e.g., large enterprise environments.

Brief Summary Text (8):
To achieve effective centralized administration of network clients, ideally a server should store a complete image of the software, including the operating system (O/S), utilized by a client computer. As each client computer is powered-on by its user, that client would establish a connection with the server and boot its operating system from the server. In particular, an operating system image would be stored on the server and would be transferred, via the network, to the client onto which that operating system would be set-up and run. Application software could then be transferred from the server, as needed, to the client and then run on the client. Proceeding in this fashion could eliminate the need for any hard disk storage on each client, thereby facilitating use of low-cost "diskless" computers. Alternatively, application software could remain on the server and be executed therefrom.

Brief Summary Text (11):
At present, in practice, during the course of configuring a particular client computer for server-based setup of the Windows 95 O/S, this O/S would create two directories on the client computer for subsequent transfer to a server: a machine directory which stores configuration information for that specific client, and a share directory which contains shared O/S files. During the boot process, both directories are used to control the remainder of the set-up process and load appropriate O/S files onto the client. It was empirically found that, from time to time--though not all the time,

while creating the directories on the server, the client computer would simply halt, i.e. "freeze". In other instances where the directories were successfully created, users found that sometimes during the network boot process, the client computer would also halt--again from time to time, though here too not always. Moreover, apart from these problems, the Windows 95 operating system, as it currently stands, apparently does not support network booting through a 32-bit local area network (LAN) adapter. While 16-bit ISA (Industry Standard Architecture) network (e.g., LAN) adapters were prevalent at the time the Windows 95 O/S was introduced, currently 32-bit adapters, particularly PCI (Peripheral Component Interconnect) type adapters predominate a network adapter market. Lastly, difficulties arose in changing a server-based installation of a Windows 95 client, thereby frustrating centralized remote administration of the application software to be downloaded to that client.

Brief Summary Text (13):
Therefore, a need exists in the art for a technique, specifically apparatus and accompanying methods, for use in a client-server environment for reliably booting an operating system, such as a 32-bit O/S, on a client PC from a network server. Moreover, such a technique, should be fully operative with all currently available LAN adapters, whether, e.g., 16- or 32-bit and permit changes to be easily made to any server-based installation of client software. Through use of such a technique, the processing load on the network server(s) can be advantageously reduced by utilizing local client RAM memory and CPU resources, rather than server resources, for client application and client O/S processing. Consequently, by utilizing these client resources, server complexity and cost, such as those occasioned by large amounts of RAM and multiple CPUs that might otherwise be required, could be sharply reduced. By virtue of meeting these needs and being capable of network booting an O/S, such as the Windows 95 O/S, on substantially any client PC, including thin-clients and diskless PCs, such a technique should find widespread use in implementing effective centralized client administration in, e.g., large networked enterprises and in providing significant administrative cost savings.

Brief Summary Text (15):
The present invention overcomes the deficiencies in the art and satisfies these needs by providing, through a network server, seamless and continuous client hard disk emulation, at a physical sectorized level, throughout the entire boot process even though, during this process, various O/S files are downloaded and O/S processes are activated which collectively change a client processing mode from real to protected and which reset and gain control of a network adapter in the client.

Brief Summary Text (16):
In accordance with my inventive teachings, a network server stores an image of a client hard disk, including the client O/S and desired applications. Rather than directing each request that arises during booting of the client O/S, for a specific sector of a stored file to a local hard disk on the client, an inventive transport driver, which is downloaded and installed on the client as part of the client O/S, redirects that request to the network server to retrieve and download that sector, from the client hard disk image, to the client. As a result, the source of the sectors remains transparent to the O/S, i.e., the client O/S is unaware that it is being booted remotely from a network server in lieu of a local hard disk drive. Each such request is processed through my inventive random access trivial file transfer protocol (RATFTP) server executing in the network server to provide read/write sectorized access to the client image file.

Brief Summary Text (17):
Specifically, whenever a user energizes a client computer (such as, e.g., a PC), this PC establishes a network connection to the network server and issues a boot request to that server. In response to this request, the network server downloads sufficient files from the stored client O/S image to the client PC to permit the client to boot the O/S and continue loading the required O/S files from that image.

Brief Summary Text (18):
During booting, the client PC initially operates in a real mode and then, based on the client O/S processes then initiating, transitions to a protected mode.

Brief Summary Text (19):
While the boot process is occurring but prior to the availability of any client O/S-based network support, client hard disk emulation occurs through appropriate calls made to an interrupt (Interrupt 13 or simply "Int 13") handler. Through such calls, appropriate sectors in the client image file are initially downloaded, via a real-mode

network adapter (NIC) driver and the Int 13 Handler to remotely install various components of the O/S into client PC. The actual client hard disk emulation process is provided through a real mode procedure that executes as part of Int 13 Handler. In essence, the real mode procedure determines, based on values of status flags, whether the client O/S is then capable of handling a network request for sector access of the client image file. If the client O/S has not then progressed to that point in its boot process, the real mode procedure processes that request, in real mode, through the Int 13 Handler.

Brief Summary Text (20):
As a client O/S kernel is installed and initialized during the boot process, the kernel installs and activates various device drivers, including the inventive LANHDVSD.VXD procedure. This procedure is compliant with both the Int 13 Handler and with the O/S, specifically, in the case of Windows 95 O/S, a network driver (NDIS--network driver interface specification) kernel therein and the O/S input/output subsystem (IOS). The inventive procedure, which executes as a protected mode driver, contains two asynchronous procedures. These asynchronous procedures, by setting and testing appropriate flags used as processing state semaphores, collectively control the transition of hard disk requests to the networked client image from the Int 13 Handler to the client O/S depending upon, as the client O/S is then booting, the O/S resources that are then available. During early phases of the boot process, insufficient O/S components have been loaded and activated to provide client O/S supported network access. Consequently, client hard disk access requests are handled through the Int 13 Handler. Whenever sufficient O/S resources become available to permit network access through the client O/S, the asynchronous procedures permit these requests to be serviced by the NDIS and IOS components of the client O/S, so as to provide O/S supported network access, rather than by the Int 13 Handler. Hence, these asynchronous procedures collectively assure, in conjunction with Int 13 Handler, seamless and continuous client hard disk emulation during the real-protected transitory state.

Brief Summary Text (21):
Inasmuch as client hard disk emulation occurs at a physical, i.e., sector, level, rather than at a higher level, my present invention, as one of its features, is independent of and properly operates with substantially any particular client file system, whether it is, e.g., FAT, FAT32, HPFS or NTFS.

Drawing Description Text (5):
FIG. 2B depicts an alternate embodiment of client hard disk image files as could be stored on hard disk 54 within server 50 shown FIG. 2A;

Drawing Description Text (8):
FIGS. 4A and 4B collectively depict, at a high-level, sequential message flow involving client PC 10 and various illustrative network servers 50 and 410 for network booting the client PC, in accordance with my present invention, and various states of the client PC along with simultaneously occurring operational status while being so booted;

Drawing Description Text (9):
FIG. 5A depicts an illustrative listing for BOOTPTAB file 500 that resides within server 50 and is used during network booting of client PC 10;

Drawing Description Text (10):
FIG. 5B depicts an illustrative listing for LANHD.INI file 550 that resides within server 50 and is also used during network booting of client PC 10;

Drawing Description Text (11):
FIG. 6 depicts a high-level block diagram of various software processes, including LANHDVSD.VXD procedure 245, that, in accordance with the present invention, collectively network boot client PC 10;

Drawing Description Text (12):
FIG. 7 depicts a start-up sequence of events, that occur in client PC 10 and in accordance with my present invention, to network boot a client operating system, and accompanying processing modes which occur during that sequence;

Detailed Description Text (2):
After considering the following description, those skilled in the art will clearly realize that the teachings of the present invention can be readily utilized in conjunction with many different computer operating systems to provide network booting

of any such O/S to a client computer. Nevertheless, to simplify the ensuing description, I will discuss my invention in the illustrative context of use with remotely installing and booting the Windows 95 O/S on a client computer in a client-server environment, where the client computer is illustratively a personal computer (PC). For ease of reference, this PC will be referred to hereinafter as a "client PC".

Detailed Description Text (3):
In this context, FIG. 1 depicts a high-level simplified block diagram of client-server environment 5 in which client PC 10 is to be booted through server 50. As shown, client PC 10 is connected, via links 20 and 40, and network 30, to network server 50. Inasmuch as the particular implementation and architecture of network 30 are both irrelevant, the ensuing discussion will omit all such details. Through the present invention, a complete image of the Windows 95 O/S that is to execute on client PC 10 is stored as image 56 on hard disk 54 within memory 52 of server 50.

Detailed Description Text (4):
In operation, whenever a user energizes (powers-up) client PC 10, this PC then establishes a network connection to the server and issues a boot request, as symbolized by line 62, to the server. In response to this request, as symbolized by line 64, the server downloads sufficient files from the stored client O/S image to the client PC to permit the client to boot the O/S and continue loading the required O/S files from the server.

Detailed Description Text (5):
In accordance with the inventive teachings, to facilitate reliable network booting of, e.g., a 32-bit client O/S, such as Windows 95 O/S, the server implements sector-by-sector hard disk emulation of a local hard disk on the client PC. The server stores an image of a client hard disk, including the O/S and desired applications. Rather than directing each request that arises during booting of the client O/S, for a specific sector of a stored file, to a local hard disk on the client, my inventive transport driver, which is downloaded and installed on the client as part of the client O/S, redirects that request to the server to retrieve and download that sector, from the client hard disk image, to the client PC. As a result, the source of the sectors remains transparent to the O/S, i.e., the client O/S is unaware that it is being booted from a network connection in lieu of a local hard disk drive. To provide reliable network booting, the inventive technique advantageously provides seamless and continuous client hard disk emulation throughout the entire boot process even though, during this process, various O/S files are downloaded and O/S processes, such as a Windows 95 network driver, are activated which collectively change the processing mode of the client PC from real to protected and which reset and gain control of a network (e.g., LAN) adapter in the client PC.

Detailed Description Text (6):
Once a processing mode is changed by the Windows 95 O/S to protected from real, that O/S would isolate a local hard disk from being directly read- or write-accessed outside of the O/S. Conventionally speaking and in the absence of using the inventive teachings, this mode change would frustrate continued server-based client hard disk emulation. Also, once the Windows 95 O/S gains control, i.e., "ownership", of the network adapter, no process external to the O/S could then gain direct access to that adapter. Consequently, again in the absence of using the inventive teachings, this too would frustrate continued server-based client hard disk emulation. As will be seen, my present invention advantageously overcomes both of these conventional limitations.

Detailed Description Text (8):
As shown, client PC 10 contains LAN adapter (also commonly referred to as a network interface card--NIC) 360. Each such NIC carries a unique physical hardware address, referred to as a media access control (MAC) address, through which that card can be uniquely addressed on a network. An illustrative MAC address is "00A024Baf9a5". Each NIC also contains internal read only memory 362 that stores boot code 364, which contains a BootP client process. Though this code is usually stored within the NIC, as shown here, this code could alternatively be implemented within a PC ROM BIOS (basic input output system) located on a motherboard of the client PC. With the boot code stored in the NIC, as shown, and read into memory of the PC on power-up and executed, the client PC establishes a network connection, through network 30 and connections 20 and 40, with remote server 50 for remotely booting of the client PC. Server 50 contains, to the extent relevant to the present invention, TCP (transmission control protocol) servers 230, specifically: either BootP server 232 or DHCP (dynamic host configuration protocol) server 234, and my inventive random access trivial file

transfer protocol (RATFTP) server 236. The BootP and DHCP servers are conventional in nature and, as such, will not be discussed in any detail. On the other hand, the RATFTP server, while based on and extends capabilities of a conventional trivial file transfer protocol (TFTP) server, accesses individual desired sector(s) (rather than just a complete file as does a conventional TFTP server), on hard disk 54 situated within server 50--thus facilitating client hard disk emulation. Such sectors are specified by a boot loader and downloaded into client PC during the network boot process.

Detailed Description Text (9):
In addition to TCP servers 230, server 50 also stores, on hard disk 54, a directory containing client PC hard disk image 56.sup.1. This image contains all O/S files 240 and accompanying user application files that are to execute on client PC 10. The O/S files include LANHDVSD.VXD file 245. This file, which will be discussed in considerable detail below, when executed during start-up of the client PC, essentially permits the client hard disk to be remotely emulated through RATFTP server 236 and so as to remotely boot the client O/S from client PC hard disk image 56.sub.1 and advantageously in a manner that seamlessly permits the emulation (and remote booting) to continue: (a) during and after the processing mode of the client PC switches from real to protected; and (b) after the O/S, as it boots, takes over ownership of NIC 360. In addition to the client image file, the server also stores, on hard disk 54, LANHD.IMG file 250 and initialization file 255, i.e., file LANHD.INI. The LANHD.IMG file contains code that properly interprets Interrupt 13 calls in the client PC for hard disk accesses during an initial boot process, i.e., prior to availability of network support through client O/S. The LANHD.INI file stores initialization information required for remote booting. This information specifies a corresponding network server, in terms of its IP (Internet protocol) address on the network, which stores a client hard disk image file for each of a group of client PCs (if not all such remotely bootable PCs) that can be remotely booted from the network and a directory on that server in which the image file is located. In particular, as shown in FIG. 5B and discussed below, the LANHD.INI file contains a series of entries, with each entry specifying a different MAC address, a corresponding server IP address therefor and a directory name. Once remote booting commences, server 50 downloads, through client hard disk emulation on a sector-by-sector basis and as symbolized by dashed line 260, image file 56.sub.1 to client PC 10 in order to remotely boot that client PC. The sequential operations that effectuate remote booting will be discussed below first in conjunction with FIG. 4.

Detailed Description Text (10):
To readily permit centralized client software administration, environment 5 also includes administrative PC 210 (which is substantially, if not totally, identical in architecture to client PC 10), which is connected, via link 215, to network 30. Once an administrator stationed at the administrative PC establishes a networked connection to server 50 and has appropriate security and file access permissions set on the server, that administrator can access, as symbolized by long-short dashed line 270, on a read and write basis, the client PC hard disk image files, such as, e.g., file 56.sub.1, stored on the server. As a result, the administrator has the ability, as desired and from one location, to open, copy, update and change the contents of any client PC image file stored on server 50 remotely from its associated client PC and without a need for that client PC to be energized. By providing such centralized client software administration for all networked clients (of which only client PC 10 is shown in FIG. 2A for simplicity), use of the present invention should markedly reduce administrative cost, particularly in large enterprise environments.

Detailed Description Text (11):
A client PC image file need not be restricted to a single file that contains a complete hard disk image and is accessible by just its associated client PC. In that regard, FIG. 2A depicts an alternate embodiment of client hard disk image files stored on server 50. Here, client disk images 56.sub.2 contain client specific image files 280, in separate directories, and a directory containing single shared client O/S files 290. Here, each client image directory stores files unique to a corresponding client PC that is to be remotely booted, e.g., directories 280.sub.1, . . . , 280.sub.n store files for client PCs 1, . . . , n, respectively, which are downloaded during the boot process or accessed thereafter. Each such directory thus has a 1:1 correspondence, as symbolized by lines 285, with its corresponding client PC. In contrast, shared directory 290 contains files, such as LANHDVSD.VXD file 245, that are common to all remotely bootable networked PCs. Hence, directory 290 has a 1:n correspondence, as symbolized by lines 295, between it and all n remotely bootable client PCs.

Detailed Description Text (14):
As shown, client PC 10 comprises input interfaces (I/F) 310, processor 320, NIC 360,

memory 330 and output interfaces 340, all conventionally interconnected by bus 350. Memory 330, which generally includes different modalities, includes illustratively random access memory (RAM) 332 for temporary data and instruction store, diskette drive(s) (not specifically shown) for exchanging information, as per user command, with floppy diskettes, and non-volatile mass store 335 that is implemented through hard disk drive(s) 334, typically magnetic in nature. Should client PC 10 be implemented by "diskless" computer, then all disk drives, including both floppy diskette drive(s) and hard disk drive(s) 334, would be omitted. Regardless of whether client PC 10 contained a hard disk drive or not, the client O/S, during its boot process, would be downloaded into RAM 332 and executed therefrom. As shown above in FIG. 2A, NIC 360 contains internal read-only memory 362, that stores network boot code 364. This code, as will be discussed shortly below, once downloaded into RAM 332 on power-up permits the NIC to establish a network connection to a remote server.

Detailed Description Text (15):
Incoming information can arise from two illustrative external sources: network supplied information, such as, in the context of network booting, sectorized information from a client hard disk emulation process executing in a network server, through network connection 20 to NIC 360, or other information from a dedicated input source--should it be connected, via path 305, to input interfaces 310. Since such a dedicated input source is not relevant here, it will not be discussed in any further detail. Suffice it to say that input interfaces 310 contain appropriate circuitry to provide necessary and corresponding electrical connections required to physically connect and interface each differing dedicated input source to client PC 10.

Detailed Description Text (18):
I will now discuss, at a high level, the sequential message flow that occurs, in accordance with my present invention, between client PC 10 and the network to remotely boot this client. FIGS. 4A and 4B collectively depict this message flow, involving client PC 10 and illustrative network servers 50 and 410; the correct alignment of these figures is shown in FIG. 4. FIGS. 4A and 4B also depict the operational states and status of the client PC during its remote booting. Since this discussion will also refer to BOOTPTAB file 500 and LANHD.INI file 550 which are respectively and illustratively shown in FIGS. 5A and 5B, then, for ease of understanding, the reader should simultaneously refer to FIGS. 4A, 4B, 5A and 5B throughout the following discussion.

Detailed Description Text (19):
Once a user has powered-up client PC 10, as symbolized by block 420, the stored ROM BIOS in the client PC is loaded into RAM 332 (see FIG. 3) of the client PC from which that code is then executed by the PC. This operational mode is denoted by block 425 shown in FIGS. 4A and 4B. At this point, as symbolized by block 430, the client PC is not aware of its IP address. The client PC then reads the boot code from a ROM situated on the NIC (or alternatively on the motherboard of the client PC itself) into RAM 332 and then executes that code--this operational mode denoted by block 450. In response to this code, the client PC will broadcast, as symbolized by line 432, a BootP (or DHCP) request packet over the network to elicit a response from a BootP (or DHCP) server. Illustratively, server 50 contains BootP server 232. This packet contains the hardware address of the NIC. For exemplary purposes, I will assume that address is "00A024Baf9a5". BootP server 232, which is a conventional TCP server, permits a network device, such as the NIC, to obtain its own IP address (i.e., here an IP address assigned to client PC 10), the name of a boot file to download, an IP address of a network server (here server 50) on which that boot file is located, and (where appropriate) an IP address of a default router. BootP server 232 does not download the boot file itself; that occurs, as will be shortly seen by TFTP server 402 executing within server 50. The IP address of the device can also be obtained through a DHCP request packet. DHCP is a newer protocol than BootP, and builds on and replaces BootP. Inasmuch as the BootP and DHCP protocols are conventional and well-known, I will not discuss them in any further detail. In that regard, for further information, the reader is referred to Chapter 19, "Booting Internet Hosts with BootP and TFTP" on pages 343-359 of P. Miller, TCP/IP Explained (.COPYRGT.1997, Digital Press)--hereinafter the "Miller" text; and Chapter 16, "BOOTP: Bootstrap Protocol" on pages 215-222 of W. R. Stevens, TCP/IP Illustrated, Volume 1--The Protocols (.COPYRGT.1994, Addison-Wesley Inc.). Both of these chapters are incorporated by reference herein. Since, for purposes of the present invention, either the BootP or DHCP protocols can be used with identical results, then, to simplify the ensuing discussion, I will omit any further reference to use of the DHCP protocol. The BootP server utilizes BOOTPTAB file 500. This file, illustratively shown in FIG. 5A, contains an entry for each of a number of remotely bootable devices that can connect to the network. Each such entry, such as entry 520

within entries 510, specifies for a single associated device: a hardware address (ha), i.e., a MAC, for that device; an associated boot file (bf) for that device; a home directory (hd) on that server which contains the boot file; and an IP address (ip) to assign to that device. For ease of access, the boot file and home directory reside on the same server as the BOOTPTAB file, here server 50. While the network may contain multiple BootP servers (of which, for simplicity, only one of which is shown in FIGS. 4A and 4B), each remotely bootable device, such as a given NIC, has only one unique corresponding entry in only one BOOTPTAB file. In this manner, a broadcast BootP request appearing on the network from a given device will engender only one response from a single server that has an entry, in its BOOTPTAB file, that contains a MAC matching that contained in the request.

Detailed Description Text (20):
In response to the BootP request packet, server 50 (also denoted as server 1), specifically BootP server 232 therein, will issue, as symbolized by line 436, a BootP reply packet onto the network. In particular, the BootP server will search through BOOTPTAB file 500 to locate an entry containing a MAC that matches that in the BootP request packet. If a match is found, as is the case for server 50, then BootP server 232 will issue the BootP reply packet. This reply packet will contain, from the parameters specified in the entry located in the BOOTPTAB file (only one such entry is illustratively shown in FIGS. 4A and 4B), an IP address assigned by the BootP server to client PC 10 (here 132.147.001.006), an IP address of server 50 (this IP address not being specifically shown in the entry) at which the boot file can be accessed, and a complete path on the server to the boot file (here .backslash.LANHD.backslash.LANHD.IMG). The boot file, here illustratively boot file 250 named LANHD.IMG, contains real-mode client hard disk emulation code as well as a name of an initialization file to be accessed. The initialization file, specifically LANHD.INI, specifies a full path, including a file name, of a client image file and a network server on which that file is stored. For illustrative purposes, the client image for client PC 10 is shown as being stored on network server 410 (also denoted as server 2) which is different from that (server 50) storing the boot file. In other situations, these two network servers could be the same.

Detailed Description Text (21):
In any event, after the client PC appropriately processes the BootP reply, the client PC will then know, as symbolized by block 440, its IP address. Next, as symbolized by line 442, the PC will issue, through the NIC, a TFTP request (typically a TFTP read command) to server 50, specifically TFTP server 402 thereon, to download the boot file identified in the BootP reply packet. If the TFTP server can locate and open this file based on the information provided in the TFTP request, then, as symbolized by line 444, TFTP server 402 will download the boot file to client PC 10. Once the boot file has been completely downloaded into RAM 332 (see FIG. 3) on client PC 10, this PC will acknowledge a successful download by issuing, as symbolized by line 446 shown in FIGS. 4A and 4B, a TFTP acknowledgement (ACK) packet, back to server 50. With the boot file (LANHD.IMG) residing, as symbolized by block 460, in the client PC and after the ACK packet is issued, the client PC will begin executing the boot file from RAM 332 to implement client hard disk emulation. At this point, client PC 10 begins operating, as symbolized by block 470, under control of the downloaded boot file (LANHD.IMG) and ceases operating under the ROM boot code previously downloaded from, e.g., the NIC.

Detailed Description Text (22):
The boot file, early in its execution, will cause the client PC to issue -a-TFPT request, as symbolized by line 462, back to server 50 to download an initialization file, specifically LANHD.INI file 550. This initialization file, as shown in FIG. 5B, also contains a series of entries. Here, each such entry, of which entry 560 is typical, contains a MAC, an IP address of a server that stores a client image file for the device having that MAC and a complete path to the client image file (here file 414 on server 410) on that server. A LANHD.INI file entry also contains an illustrative term, such as "3c90x" or "3c5x9", which merely describes a name of the NIC associated with that entry and is ignored, as a comment field, during subsequent processing of this file by the client PC. A further parameter (db) in entry 560 defines a default boot option (illustratively set to A or C) which is not relevant here. Once the download, as symbolized by line 464, completes, client PC 10 will then generate and transmit, as symbolized by line 466, a TFTP ACK packet, over the network back to server 50. Furthermore, once this file has been downloaded, the client PC, under control of the boot file, LANHD.IMG, will process this file by first checking the contents of this file to determine whether an entry in the file contains a MAC that matches that of the NIC in the client PC. When an entry having a matching MAC is found, as illustratively occurs here, the boot file will then extract, from that entry, the full path to the

client image file (here C: .backslash.LANHD.backslash.disk150) and the IP address (here 132.147.001.001) of a network server (here server 410) on which the client image file resides. Once client PC 10 obtains this information from the initialization file, the client PC, specifically executing the boot file (LANHD.IMG) which is then performing client hard disk emulation, issues, as symbolized by line 475, an RATFTP request to network server 410 to download a boot sector from client PC image file 414 residing thereon.

Detailed Description Text (23):
In response to the RATFTP request, RATFTP server 412 executing on network server 410 downloads the boot sector from the client PC image file to client PC 10 and specifically into RAM 332 (see FIG. 3) on that PC. Once this boot sector is completely downloaded into RAM, the client PC then, as symbolized by block 480, executes it. Under control of the boot sector, various remaining files of the 32-bit client PC O/S are downloaded in sequence and their execution started in order to initiate associated O/S processes and, by doing so, complete the start-up of the client O/S on PC 10. Such downloading occurs on a sector-by-sector basis through corresponding RATFTP read operations. In addition, the client O/S can write information, also on a sector-by-sector basis (through client hard disk emulation), into its client image file through RATFTP write operations. These RATFTP operations are symbolized by line 484 and are preceded by a request, as symbolized by line 482, to download/upload information between server 410 and client PC 10. Each such RATFTP operation is acknowledged, as symbolized by line 486, upon its completion, by the device that receives data, i.e., either the client PC or network server 410. From the point that the boot sector takes over control of the client PC, the PC is thereafter operated by the client O/S, as symbolized by block 490, rather than by the boot file LANHD.IMG.

Detailed Description Text (24):
FIG. 6 depicts a high-level block diagram of various software processes, including LANHDVSD.VXD procedure 245, which executing in the client PC collectively network boot that PC. To fully appreciate the interaction of these procedures, the reader should also simultaneously refer throughout the following discussion to FIG. 7 which shows basic start-up sequence 710 that occurs within the client PC for implementing network booting and the associated processing modes 750 that simultaneously occur in that PC.

Detailed Description Text (25):
During network booting, the client PC initially operates in a real processing mode and then, based on the client O/S processes then initiating, transitions to a protected processing mode. LANHDVSD.VXD procedure 245 permits client PC hard disk emulation to seamlessly continue while the processing modes change during O/S booting as well as after the client O/S takes ownership of the NIC in the client PC.

Detailed Description Text (26):
During the boot process and prior to the availability of any client O/S-based network support, client hard disk emulation occurs, as discussed above in conjunction with LANHD.IMG operation shown in FIGS. 4A and 4B, through appropriate calls made to Interrupt 13 (Int 13) handler 623. Through such calls, appropriate sectors in the client image file are downloaded, via real-mode NIC driver 625 and Int 13 Handler 623 to remotely install various components of O/S 690 into client PC. The actual client hard disk emulation process is provided through Real Mode Procedure 900 that executes as part of Int 13 Handler 623. In essence, procedure 900 determines, based on values of status flags, whether the client O/S is then capable of handling a network request for sector access of the client image file. If the client O/S has not then progressed to that point in its boot process, procedure 900 processes that request, in real mode, through Int 13. The remainder of this handler is conventional in nature.

Detailed Description Text (27):
In particular, as shown in start-up sequence 710, upon power-up of the client PC, this PC commences operation using real-mode processing, as symbolized by line 752. This mode of operation persists through downloading and initiation of the boot loader and client hard disk emulation code, i.e., boot file LANHD.IMG; and commencement of loading 32-bit client O/S 690 (e.g., Windows 95 O/S). Here, client hard disk emulation with sector-by-sector downloading is provided by block 620, specifically Int 13 Handler 623 and real-mode NIC driver 625.

Detailed Description Text (28):
Once a file, e.g., Windows 95 file "Win.com", which loads the kernel of the client O/S (i.e., Windows 95 file "VMM32.VXD") has fully loaded and its execution has then been initiated to actually load the O/S kernel, the processing mode of the client PC begins

transitioning between real and protected, and continues, as symbolized by line 754, in this transitory state until an Input/Output sub-system (IOS) component within the client O/S has itself initialized. Once this kernel is installed by file "Win.com" and initializes, the kernel then installs and activates various device drivers. LANHDVSD.VXD procedure 245 is one of these drivers. This procedure is compliant with both Int 13 Handler 623 and with the O/S, specifically a network driver (NDIS--network driver interface specification) kernel therein and the IOS. Procedure 245 assures that client hard disk emulation occurs either through the Int 13 Handler or the client O/S, specifically an O/S NDIS (network driver interface specification) transport driver therein, based on the processing state and O/S boot status of the client PC. As shown, procedure 245, which executes as a protected mode driver, contains start-up procedure 800, asynchronous procedures: NDIS check 1000 and Protected Mode Helper 1100, IOS vendor supplied driver 642, and NDIS transport driver 655. The two asynchronous procedures collectively provide, in conjunction with Int 13 Handler 623, seamless client hard disk emulation during the real-protected transitory state. These asynchronous processes, through setting and testing appropriate flags (specifically "NDIS Loaded" and "Need Protected Mode Helper" flags, as discussed in detail below) used as processing state semaphores, collectively control the transition of hard disk requests to the networked client image from the Int 13 Handler to the client O/S depending upon, as the client O/S is then booting, the O/S resources that are then available. In that regard, when sufficient O/S resources become available to permit network access through the client O/S, then the asynchronous procedures permit these requests to be serviced by the NDIS and IOS components of the client O/S rather than by Int 13 Handler 623.

Detailed Description Text (29):
IOS vendor supplied driver 642 binds itself to one of the thirty-two abstraction layers situated within IOS 680. Once this occurs, driver 642 then appropriately handles client hard disk requests that originate with the O/S based on the processing mode of the client at the time: either by effectively invoking Int 13 Handler to process the request (for non-O/S supported network access) or sending the request to the NDIS processes in client O/S (for O/S supported network access).

Detailed Description Text (30):
Specifically, once LANHDVSD.VXD procedure 245 is loaded during the real-protected transitory state, this procedure executes start-up procedure 800. Also, during this state, the client O/S activates other drivers as well. One such driver is NDIS transport driver 655. This driver is a protocol stack that communicates with the NIC through NDIS rather than directly (as occurs with the Int 13 Handler). Since a finite period of time will be needed for an NDIS kernel and the NDIS transport driver to fully activate and this transport driver to bind itself to the NDIS kernel and to IOS 680, start-up procedure 800 will launch Asynchronous Procedure: NDIS check 1000 (which will be discussed in detail below in conjunction with FIG. 10). In essence, once launched, NDIS check procedure 1000 then regularly determines whether an NDIS kernel (here NDIS kernel 660) of the client O/S is then executing. If it is executing, the client O/S will have already switched to the protected mode. Hence, procedure 1000 sets an appropriate status flag to signify that the O/S has entered this mode, as symbolized by line 756, and launches Asynchronous Procedure: Protected Mode Helper 1100. This latter procedure, should the current state of the client-O/S permit, will service any subsequent Int 13 requests for hard disk access through the client O/S, specifically NDIS. Thus, all subsequently accessed client image sectors, occurring in protected mode, will be accommodated through IOS 680; O/S-based network support block 640, specifically through IOS vendor supplied driver 642, NDIS transport driver 655, NDIS kernel 660, and protected-mode NIC driver 673; and NIC 360. Inasmuch as IOS 680, NDIS transport driver 655, NDIS kernel 660, IOS vendor supplied driver 642 and NIC drivers 625 and 673 are all conventional, I will not discuss these in any further detail.

Detailed Description Text (32):
In particular, upon entry into procedure 800, execution first proceeds to decision block 810. This decision block, when executed, tests whether Int 13 Handler 623 has been installed. If this handler has not yet been installed, then client PC hard disk emulation can not occur. Consequently, execution simply exits from procedure 800, via NO path 813. Alternatively, if this handler is installed, then decision block 810 directs execution, via YES path 815, to block 820. This latter block inserts an entry, as a vendor supplied driver, for procedure 245 into an appropriate layer in IOS 680. This entry, which will be specifically used by IOS vendor supplied driver 642 (as discussed above), is used as an entry point for client hard disk requests. Once this entry point has been created, then execution proceeds to block 830. This block, when executed, binds LANHDVSD.VXD procedure 245, specifically NDIS transport driver 655

therein (see FIG. 6), as a transport driver to NDIS kernel 660. Once this occurs, procedure 800 then executes block 840 which, in turn, starts Asynchronous Procedure: NDIS check 1000. Once this procedure 1000 is started, execution exits from start-up procedure 800.

Detailed Description Text (33):
FIG. 9 depicts a flowchart of Real Mode Procedure 900 that executes within the client PC and specifically within Int 13 handler 623. As noted above, procedure 900 determines, based on values of status flags, whether the client O/S is then capable of handling a network request for sector access of the client image file. If the client O/S has not then progressed to that point in its boot process, procedure 900 processes that request, in real mode, through Int 13.

Detailed Description Text (34):
In particular, procedure 900 executes at the occurrence of an Int 13 request for a client hard disk access. Such a request specifies a particular drive, sector, cylinder, head and buffer from which stored data is to be read. Upon entry into procedure 900, execution first proceeds to decision block 910. This block, when executed, determines whether an "NDIS Loaded" flag has been set to one, thereby signifying that the NDIS portion of the client O/S has been loaded and is active. If this portion either is not loaded or is not yet fully activated, then decision block 910 routes execution, via NO path 913, to block 920. This latter block, when executed, attempts to service the hard disk request through the Int 13 Handler. Thereafter, execution proceeds to decision block 930 to test whether this attempt was successful. If the request has been successfully serviced over the network through the Int 13 Handler, i.e., the request did not fail, then execution exits from procedure 900, via NO path 933, with the results of that request then being returned.

Detailed Description Text (35):
Alternatively, if the request failed, then decision block 930 routes execution, via YES path 935, to decision block 940. This latter decision block also tests the status of the "NDIS Loaded" flag to determine whether the NDIS portion of the client O/S has now been loaded and is now active. If the NDIS portion either has not been loaded or is not fully active, then execution is fed back, via NO path 943, back to this decision block to await availability of the NDIS portion. If, however, the NDIS portion is loaded and active, then decision block 940 routes execution, via YES path 945, to block 950. This latter block sets the value of a "Need Protected Mode Helper" flag to one. Execution is also directed to block 950 in the event decision block 910 determined that the value of the "NDIS Loaded" flag is set to one, i.e., the NDIS portion is fully active upon entry into procedure 900. Setting the "Need Protected Mode Helper" flag to one at this point indicates that although an Int 13 request is pending, that request requires protected mode processing. Such a need can arise where at the time the request was issued insufficient NDIS resources were active, but subsequently and before the request was handled, those resources became active (by virtue of various NDIS processes having been started, bound and initialized in the interim). Once block 950 sets the value of the "Need Protected Mode Helper" flag, execution proceeds to decision block 960. This latter block tests whether this flag (serving as a semaphore) has been reset to zero by another procedure, i.e., Asynchronous Procedure: Protected Mode Helper 1100, thereby signifying that real mode client hard disk emulation is currently required. As long as the value of the flag remains at one, indicating that protected mode hard disk emulation is to occur through LANHDVSD.VXD, execution will loop back to the decision block to effectively poll this flag, on a routine basis, until its value changes. Alternatively, if the value of the flag is set to zero, hence indicating that real mode disk emulation is needed to handle the current Int 13 request, then execution exits, via YES path 965, from procedure 900 with results of that request, as produced by the Int 13 Handler, then being obtained and returned.

Detailed Description Text (38):
FIG. 11 depicts a flowchart of Asynchronous Procedure: Protected Mode Helper 1100. As noted above, this procedure, should the current state of the client O/S then being booted permit, will service any subsequent Int 13 requests for hard disk access through the client O/S, specifically NDIS.

Detailed Description Text (40):
Alternatively, if an IOS request has not been issued for the current access request, then decision block 1120 routes execution, via NO path 1125, to decision block 1130. This latter decision block, when executed, tests whether the "Need Protected Mode Helper" flag is set to one. If this flag has been so set, such as through real mode procedure 900, this indicates that O/S boot has sufficiently progressed to the point

where client hard disk emulation needs to occur on a protected mode basis with O/S support through LANHDVSD.VXD. In this instance, decision block 1130 will route execution, via YES path 1133, to block 1140. This latter block, when executed, will service the pending Int 13 request via a RATFTP call through NDIS, effectively providing O/S supported network access. Such an Int 13 request would be pending at this point inasmuch as this flag would have been previously set to one by Real Mode Procedure 900 in response to such a request and if either the NDIS portions of the client O/S, were, during execution of procedure 900, not sufficiently active, or became sufficiently active after routine 900 was just entered. As discussed above, if the "Need Protected Mode Helper" flag is set to one, this indicates that although an Int 13 request is now pending, that request requires protected mode processing. Such a need can arise where at the time the request was issued insufficient NDIS resources were active, but subsequently and before the request was handled, those resources became active (by virtue of various NDIS processes having been started, bound and initialized in the interim). Thereafter, once this Int 13 request is processed, via execution of block 1140 and through NDIS, execution proceeds to block 1150 which, in turn, resets the value of the "Need Protected Mode Helper" flag to zero. Doing so precludes any processing of further Int 13 requests through NDIS. Once block 1150 completes its execution, execution is directed, via paths 1155 and 1160 back to block 1110 to await a next IOS request, and so forth. Execution is also directed to path 1160, via NO path 1135, in the event decision block 1130 determines that the "Need Protected Mode Helper" flag is zero.

Detailed Description Text (41):
As discussed above, my inventive RATFTP server resides on the network server which stores the client PC image file. The RATFTP (random access trivial file transfer protocol) server permits that image file to be accessed on a random access sectorized basis as would be required by the client O/S if that client were to be booted from its local hard disk.

Detailed Description Text (48):
By virtue of downloading desired physical sectors of a client PC hard disk image file from a network server to a client computer, hence operating at a physical level of the disk itself, the present invention is independent of and will properly operate with substantially any client file system, whether it is, e.g., FAT, FAT32, HPFS or NTFS.

Detailed Description Text (49):
Furthermore, though I have described my invention in conjunction with use with Windows 95 O/S, those skilled in the art will realize that the teachings of my invention are applicable to use with nearly any client O/S, such as, e.g., 32-bit operating systems, to seamlessly emulate a local hard disk and perform network booting while that O/S, during its boot process, transitions from real to protected processing mode and/or gains ownership of a network adapter in the client.

Other Reference Publication (1):
P. Miller, TCP/IP Explained (.COPYRGT. 1997, Digital Press), specifically Chapter 9, "Booting Internet Hosts with BootP and TFTP", pp. 343-359.

CLAIMS:

1. A method of booting an operating system (O/S) to a client computer from a network, the network storing an image on a first server, the image having O/S files for the client computer, the method comprising the steps, in the client computer, of:

(a) issuing a corresponding request to the first server to download contents of a corresponding one of each of a plurality of sectors residing on the first server so as to form a plurality of requests, wherein:

(a1) said sectors collectively store an image file which contains the image; and

(a2) the first server emulates, in response to said plurality of requests, behavior of a disk drive on the client computer such that contents of individual physical sectors of the image file specified in corresponding ones of the requests are accessed by the first server on a sector-by-sector basis from a disk drive associated with the first server and downloaded, via the network, to the client computer, and wherein the corresponding request issuing step comprises the steps of:

(a3) generating a request, to the first server, to download a boot sector contained in the image file; and

(a4) executing the boot sector, once the boot sector is downloaded from the first server, so as to subsequently generate each of said plurality of requests such that the client computer receives, on a sector-by-sector basis from the first server, the contents of the individual physical sectors that collectively comprise client O/S files;

(b) storing the contents of each of said sectors received from the first server; and

(c) activating client O/S processes embodied by the contents of said sectors stored on the client computer.

3. The method in claim 2 further comprising the step, in the client computer, of obtaining a boot file from a second network server, the boot file comprising real-mode client disk emulation code; and wherein the request generating step comprises the step of producing the boot sector download request through executing the boot file, wherein the boot sector request contains an identification of the first server and the name of the image file.

8. The method in claim 7 wherein the controlling step comprises the step of controlling the changeover based on whether sufficient active resources then exist in client O/S as it is booting.

10. The method in claim 7 further comprising the steps, in the client computer, of:

executing predefined boot code stored in the client computer which, in response thereto, broadcasts a client IP (Internet protocol) request message on the network, said client IP request message containing an address of the network interface situated in the client computer;

obtaining, from the second server, a reply message to the client IP request message, wherein the reply message contains an IP address assigned to the client computer, an IP address of the first server and an address of the boot file stored on the second server, wherein the boot file contains the real-mode client disk emulation code and a name of an initialization file;

downloading the boot file from the second server;

executing the emulation code contained in the boot file, once it has been downloaded, so as to initiate real-mode client disk emulation through the second server and thereafter to download the initialization file from the second server, wherein the initialization file contains an address of the image file residing on the first server and an address of the first server; and

downloading, from the first server and in response to execution of the boot file in conjunction with the initialization file, a boot sector contained within the image file;

issuing, as a result of executing the boot sector, each of said plurality of requests, to the second server, to download the contents of the plurality of corresponding sectors that collectively store the image file; and

starting execution of various client O/S processes in the client computer as sufficient O/S files are downloaded on a sector-by-sector basis from the image file stored on the first server.

12. The method in claim 10 wherein the controlling step comprises the step of controlling the changeover based on whether sufficient active resources then exist in client O/S as it is booting.

15. The method in claim 5 further comprising the steps, in the client computer, of:

executing predefined boot code stored in the client computer which, in response thereto, broadcasts a client IP (Internet protocol) request message on the network, said client IP request message containing an address of the network interface situated in the client computer;

obtaining, from the second server, a reply message to the client IP request message, wherein the reply message contains an IP address assigned to the client computer, an IP

address of the first server and an address of the boot file stored on the second server, wherein the boot file contains the real-mode client disk emulation code and a name of an initialization file;

downloading the boot file from the second server;

executing the emulation code contained in the boot file, once it has been downloaded, so as to initiate real-mode client disk emulation through the second server and thereafter to download the initialization file from the second server, wherein the initialization file contains an address of the image file residing on the first server and an address of the first server; and

downloading, from the first server and in response to execution of the boot file in conjunction with the initialization file, a boot sector contained within the image file;

issuing, as a result of executing the boot sector, each of said plurality of requests, to the second server, to download the contents of the plurality of corresponding sectors that collectively store the image file; and

starting execution of various client O/S processes in the client computer as sufficient O/S files are downloaded on a sector-by-sector basis from the image file stored on the first server.

18. Apparatus for booting an operating system (O/S) to a client computer from a network, the network storing an image on a first server, the image having O/S files for the client computer, the apparatus comprising a client computer having;

(a) a processor; and

(b) a memory, connected to the processor, for storing executable computer instructions therein; and

(c) wherein the processor, in response to the instructions:

(c1) issues a corresponding request to the first server to download contents of a corresponding one of each of a plurality of sectors residing on the first server so as to form a plurality of requests, wherein:

(c1a) said sectors collectively store an image file which contains the image; and

(c1b) the first server emulates, in response to said plurality of requests, behavior of a disk drive on the client computer such that contents of individual physical sectors of the image file specified in corresponding ones of the requests are accessed by the first server on a sector-by-sector basis from a disk drive associated with the first server and downloaded, via the network, to the client computer, and wherein the processor further:

(c1c) generates a request, to the first server, to download a boot sector contained in the image file; and

(c1d) executes the boot sector, once the boot sector is downloaded from the first server, so as to subsequently generate each of said plurality of requests such that the client computer receives, on a sector-by-sector basis from the first server, the contents of the individual physical sectors that collectively comprise client O/S files;

(c2) stores the contents of each of said sectors received from the first server; and

(c3) activates client O/S processes embodied by the contents of said sectors stored on the client computer.

20. The apparatus in claim 19 wherein the processor, in response to the executable instructions:

obtains a boot file from a second network server, the boot file comprising real-mode client disk emulation code; and

produces the boot sector download request through executing the boot file, wherein the

boot sector request contains an identification of the first server and the name of the image file.

27. The apparatus in claim 25 wherein the processor, in response to the executable instructions, controls the changeover based on whether sufficient active resources then exist in client O/S as it is booting.

29. The apparatus in claim 25 wherein the processor, in response to the executable instructions:

executes predefined boot code stored in the client computer which, in response thereto, broadcasts a client IP (Internet protocol) request message on the network, said client IP request message containing an address of the network interface situated in the client computer;

obtains, from the second server, a reply message to the client IP request message, wherein the reply message contains an IP address assigned to the client computer, an IP address of the first server and an address of the boot file stored on the second server, wherein the boot file contains the real-mode client disk emulation code and a name of an initialization file;

downloads the boot file from the second server;

executes the emulation code contained in the boot file, once it has been downloaded, so as to initiate real-mode client disk emulation through the second server and thereafter to download the initialization file from the second server, wherein the initialization file contains an address of the image file residing on the first server and an address of the first server; and

downloads, from the first server and in response to execution of the boot file in conjunction with the initialization file, a boot sector contained within the image file;

issues, as a result of executing the boot sector, each of said plurality of requests, to the second server, to download the contents of the plurality of corresponding sectors that collectively store the image file; and

starts execution of various client O/S processes in the client computer as sufficient O/S files are downloaded on a sector-by-sector basis from the image file stored on the first server.

32. The apparatus in claim 29 wherein the processor, in response to the executable instructions, controls the changeover based on whether sufficient active resources then exist in client O/S as it is booting.

36. The apparatus in claim 23 wherein the processor, in response to the executable instructions:

executes predefined boot code stored in the client computer which, in response thereto, broadcasts a client IP (Internet protocol) request message on the network, said client IP request message containing an address of the network interface situated in the client computer;

obtains, from the second server, a reply message to the client IP request message, wherein the reply message contains an IP address assigned to the client computer, an IP address of the first server and an address of the boot file stored on the second server, wherein the boot file contains the real-mode client disk emulation code and a name of an initialization file;

downloads the boot file from the second server;

executes the emulation code contained in the boot file, once it has been downloaded, so as to initiate real-mode client disk emulation through the second server and thereafter to download the initialization file from the second server, wherein the initialization file contains an address of the image file residing on the first server and an address of the first server; and

downloads, from the first server and in response to execution of the boot file in conjunction with the initialization file, a boot sector contained within the image

file;

issues, as a result of executing the boot sector, each of said plurality of requests, to the second server, to download the contents of the plurality of corresponding sectors that collectively store the image file; and

starts execution of various client O/S processes in the client computer as sufficient O/S files are downloaded on a sector-by-sector basis from the image file stored on the first server.